

# Applying AUTOSAR in Practice

*Available Development Tools and Migration Paths*

---

*Master Thesis, Computer Science*

*Authors:*

Jesper Melin (jmn06007@student.mdh.se)  
Daniel Boström (dbm06002@student.mdh.se)

*Supervisor/Examiner:*

Mats Björkman  
mats.bjorkman@mdh.se

## **Abstract**

With the increased use of Electronic Control Units (ECUs) in the automotive industry, the system creation and integration becomes progressively more complex. In order to manage this issue, the AUTomotive Open System Architecture (AUTOSAR) was initiated by several of the larger automotive companies and suppliers. Their goal was to create an architecture which will increase the reusability of software, integration of solutions from multiple suppliers and improve scalability. This thesis is made in collaboration with the company CrossControl and covers questions which are specific to their interests, features the AUTOSAR standard has to offer, evaluation of the use of development tools from different vendors, how migration is supported and how the standard affects the required hardware. Among our conclusions is that the AUTOSAR goals of decoupling software application components from the hardware is met to a high degree. Secondly even though file formats are standardized it is not a seamless integration between development tools. Finally, the choice of hardware is not only affected by properties of the standard, but the current versions of tools also limit the choices.

**Keywords:** AUTOSAR, development tools, migration, trade-offs, ArcCore, Arctic Studio, OpenSynergy, COQOS.

## **Acknowledgments**

We want to thank the following people for their support and help during our thesis:

- Rikard Land at CrossControl for his experience and great help.
- Michel Svenstam at ArcCore for his knowledge and help about AUTOSAR.
- Niclas Ericson at ArcCore for the help with Arctic Studio.
- Jonas Carlsson at ArcCore for the help with hardware related issues.
- Carsten Krüger at OpenSynergy for his help with COQOS-tP.
- Michael Ummenhofer at IXXAT for the help with the USB-to-CAN adapter.

## Abbreviations and Terms

AAL	Artop AUTOSAR Layer
AL	Application Layer
API	Application Programming Interface
ARXML	AUTOSAR XML
ASIL	Automotive Safety Integrity Level
AUTOSAR	AUTomotive Open System Architecture
BSW	Basic Software
CAN	Controller Area Network
CanIf	Can Interface
CanNm	CAN Network Management
CanSm	CAN Schedule Manager
CanTp	CAN Transport Layer
CanTrcv	CAN Transceiver
CDL	Complex Driver Layer
COM	Communication
DEM	Diagnostics Event Manager
DET	Development Error Tracer
DIO	Digital Input Output
DLC	Data Length Code
E/E	Electrical and Electronics
EB	Electrobit
ECI	Embedded Card Interface
ECL	Artop Eclipse Complementary Layer
ECU	Electronic Control Unit
ECUAL	ECU Abstraction Layer
EcuC	ECU Configuration
ELF	Executable and Linkable Format
FAQ	Frequently Asked Questions
FIFO	First In First Out
GCC	GNU Compiler Collection
GDB	GNU Debugger
GPL	General Public License
GUI	Graphical user interface
HMI	Human Machine Interaction
HRH	Hardware Receive Handle
HTH	Hardware Transmit Handle
HW	Hardware
I/O	Input/Output
IDE	Integrated development environment
ISO	International Organization for Standardization
JDK	Java Development Kit
JTAG	Joint Test Action Group
LCD	Liquid crystal display

LED	Light-emitting diode
MCAL	Microcontroller Abstraction Layer
Mcu	Microcontroller
MISRA	The Motor Industry Software Reliability Association
OEM	Original Equipment Manufacturer
OMG	Object Management Group
OS	Operating System
PDU	Protocol Data Unit
PduR	Pdu Router
PWM	Pulse-Width Modulation
RAM	Random Access Memory
RTE	Runtime Environment
SCHM	BSW Scheduler module
SDG	Special Data Group
SL	Service Layer
SPEM	Software Process Engineering meta-model
SPI	Serial Peripheral Interface
SW	Software
SWC	Software Component
SWD	Serial Wire Debug
TCP	Transmission control protocol
UDP	User Datagram protocol
UML	Unified Modelling Language
VFB	Virtual Functional Bus
XML	eXtensible Markup Language

## Table of Contents

1	Introduction.....	1
1.1	Background.....	1
1.2	High-level Questions.....	1
1.3	Answers .....	3
1.4	Thesis outline.....	3
1.5	Process steps .....	3
2	The AUTOSAR standard .....	4
2.1	Background.....	4
2.2	Purpose with AUTOSAR.....	4
2.3	Common applications using AUTOSAR.....	9
2.4	AUTOSAR architecture .....	10
2.5	Software components .....	13
2.6	Methodology .....	21
2.7	Trade-offs .....	23
3	Dependability and Safety .....	25
3.1	Critical systems.....	25
3.2	Dependability .....	25
3.3	AUTOSAR and Safety .....	26
4	Survey of Development tools for AUTOSAR.....	28
4.1	Tool vendors.....	28
4.2	Artop.....	34
5	Selection of Development Tools.....	36
5.1	First iteration .....	36
5.2	Second iteration .....	39
5.3	Result from first and second iteration .....	40
6	Demonstrator .....	42
6.1	Requirements of the demonstrator .....	42
6.2	What we have done .....	42
6.3	Help from suppliers .....	43
6.4	Design decisions .....	44
6.5	Communication .....	45
6.6	Workflow .....	46
6.7	Equipment .....	47
6.8	Nodes.....	47
6.9	System .....	48
6.10	Topology cases .....	51
6.11	Mapping of groups to nodes .....	55
6.12	I/O Simulator .....	55
6.13	CAN communication with STM32 .....	60
6.14	CAN communication with COQOS-tP .....	60
6.15	Debugging.....	69
7	Evaluation.....	71
7.1	Hardware independence .....	71
7.2	Maturity of the tools .....	74
7.3	Possibility to Share AUTOSAR files within and between tools .....	76

7.4	Trade-offs in practice .....	82
8	Lessons learned and Reflections .....	84
8.1	Lesson 1: The Workflow could be smoother .....	84
8.2	Lesson 2: The Learning curve is steep .....	84
8.3	Lesson 3: Sharing files between tools .....	88
8.4	Reflections on a good AUTOSAR development tool .....	90
8.5	Reflection on Configuration and Implementation .....	92
8.6	Reflection on simulation possibilities.....	93
8.7	Reflection on suitable CrossControl products to be used with AUTOSAR .....	93
9	Conclusions.....	95
9.1	The standard.....	95
9.2	The tools .....	95
9.3	Migration .....	95
9.4	Future work .....	95
10	References.....	96
11	Appendix 1: Demonstrator ARXML models.....	101
11.1	HeatRegulator.arxml .....	101
11.2	SeatHeater.arxml.....	102
11.3	SeatHeatingController.arxml.....	103
11.4	SeatSensor.arxml.....	106
11.5	DatatypesAndInterfaces.arxml.....	107
11.6	SeatHeaterComposition.arxml .....	109
12	Appendix 2: Demonstrator software components implementations .....	111
12.1	HeatRegulator.c.....	111
12.2	SeatHeater.c .....	112
12.3	SeatHeatingController.c .....	113
12.4	SeatSensor.c .....	116
13	Appendix 3: CAN sequence diagrams.....	118
13.1	Transmit request .....	118
13.2	Transmit confirmation.....	119
13.3	Receive indication .....	120
14	Appendix 4: IXXAT ECI state machine.....	121
15	Appendix 5: Linux I/O Simulator source code .....	122
16	Appendix 6: STM32 button source code .....	124
16.1	stm3210e_eval.h .....	124
16.2	stm3210e_eval.c .....	124
17	Appendix 7: Settings for Arctic Studio modules to enable CAN.....	127

# 1 Introduction

Today's automotive industry is faced with a growing demand for technical appliances and this requires an increased use of ECUs which is reflected in the complexity of the system. Traditionally, solutions have been specific for a certain platform or model and this structure is more becoming unmanageable and costly. Instead of making specific solutions a standardized future is the way to go. This increased complexity could be manageable and improved by a standardized architecture and the solution is AUTOSAR.

This thesis will present the key elements of the AUTOSAR standard and evaluate to what extent it holds true in practice by developing a demonstrator using AUTOSAR dedicated tools.

## 1.1 Background

AUTOSAR emerged as a solution for standardized automotive software in the automotive industry but has recently started to gain interest in other industries as well thanks to its many benefits.

CrossControl is a company with a background in the forestry machine industry. They provide expertise and solutions to a broad range of industrial vehicle applications and business.

Since there is a reasonable chance future customers of CrossControl will inquire about AUTOSAR based solutions, one goal of this thesis is to research if and how AUTOSAR can be applied to CrossControl's current products.

## 1.2 High-level Questions

This thesis is made in collaboration with CrossControl and with their interest in AUTOSAR; some questions are therefore more related to their importance and others to persons who are interested to know more about the standard. Three different topics have been the centre for the question formulation: The Standard, Development Tools and Migration from existing solutions, see Figure 1.

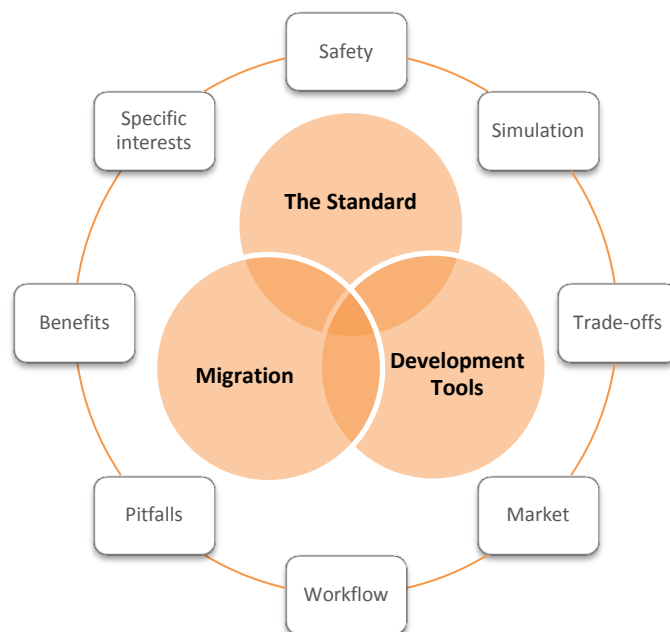


Figure 1: The three main topics with subjects

### **1.2.1 The Standard**

These questions are formulated in order to answer questions related to the standard and what it is like to develop towards the standard.

- Q1.1. What is the AUTOSAR standard and why is it created?
- Q1.2. Is it manageable to understand the whole standard or are some parts addressed for certain roles?
- Q1.3. What parts of the standard is good to know about depending on the user's role?
- Q1.4. What is it like to develop towards the standard?
- Q1.5. Has the development of the standard introduced any trade-offs?
- Q1.6. What are typical AUTOSAR applications?
- Q1.7. How does the standard support simulation?
- Q1.8. How does the standard support migration from existing solutions?
- Q1.9. How does the standard support reusability?
- Q1.10. Is it possible to reuse software components?
- Q1.11. Is it possible to reuse configuration of the Basic Software Modules?
- Q1.12. How does the standard address safety issues?
- Q1.13. Does the standard propose something which in practice is limited?

### **1.2.2 Development Tools**

These questions are asked in order to determine what it is like to work with development tools for AUTOSAR.

- Q2.1. How does the maturity of the tools look like, are they established?
- Q2.2. How is the workflow between tools from different vendors, encouraged or not recommended?
- Q2.3. Is it possible to share and reuse configurations between tools from different vendors?
- Q2.4. Is it possible to share and reuse configurations between tools from the same vendor?
- Q2.5. Development tool toolchain, is complete better than divided?
- Q2.6. How important is it that the development tools support the target hardware?

### **1.2.3 Migration from existing solutions**

The following questions are formulated in order to answer questions related to migration from existing solutions to the AUTOSAR standard.

- Q3.1. Why migrate to the AUTOSAR standard?
- Q3.2. Are any of CrossControl's products suitable to be used with the AUTOSAR standard?
- Q3.3. After a migration, how could the expected productivity and cost figures look like?
- Q3.4. Does a migration affect the requirements on the hardware?
- Q3.5. Are there any major pitfalls while migrating? If so, how can they be avoided?
- Q3.6. Is it possible to integrate the AUTOSAR methodology into an already existing workflow?
- Q3.7. How does the learning curve look like?
- Q3.8. What are the benefits with AUTOSAR compared to non-standard solutions?
- Q3.9. Is the development paradigm shifted from implementation towards configuration?
- Q3.10. After a migration, is most time spent on Configuration or Implementation?



### 1.3 Answers

Throughout this thesis, the questions will be answered using the following format:

***This answers QX.X “Question formulation”***

Answer to the question

### 1.4 Thesis outline

Chapter 2 gives an introduction to the AUTOSAR standard with the intention of familiarizing the reader with what AUTOSAR is about and its vocabulary which will be used throughout the report. Chapter 3 briefly touches upon dependability and safety aspects that are worth considering in AUTOSAR’s areas of use. Chapter 4 presents a number of tool suites discovered during the research phase and chapter 5 covers the method and the result of comparing and eventually selecting tool suites used in later chapters. Chapter 6 makes use of the selected tools in chapter 5 in order to realize a demonstrator which can be used to answer questions raised in the introduction. In chapter 7 an evaluation based on the experience gained realizing the demonstrator in chapter 6 is presented. Chapter 8 dives into lessons that were learned during this thesis and reflections that arose afterwards. Finally chapter 9 covers the conclusions that can be drawn based on the knowledge obtained from the different process steps involved in this thesis.

### 1.5 Process steps

A number of process steps were defined in a certain order according to Figure 2; these steps were required in order to be able to answer the formulated questions.

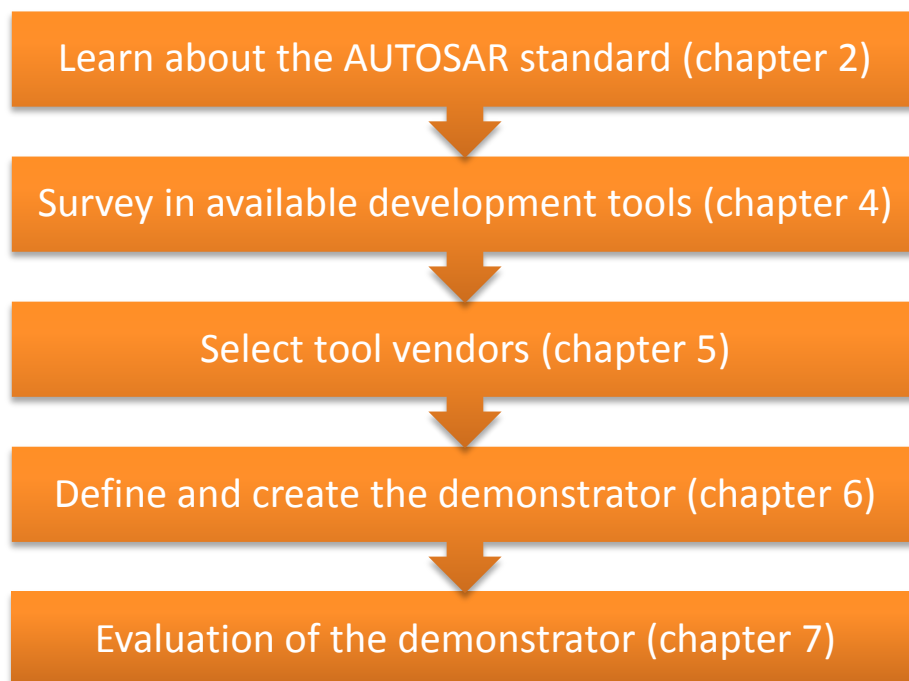


Figure 2: The thesis divided into several process steps.

## 2 The AUTOSAR standard

The following section will present the basics and fundamental parts of the AUTOSAR standard.

### 2.1 Background

Today's vehicles that are rolling out from the factories are basically computers with wheels. A modern vehicle consists of approximately 50-60 ECUs and the numbers are increasing for each year [1]. Each unit is in charge of a specific functionality and they communicate with each other over some kind of bus, e.g. CAN, FlexRay, LIN etc. For each additional unit that is connected to the system, the system complexity increases. Applications and software must be configured for each system and with new hardware the applications must be rewritten each time to support any changes in the hardware.

In order to handle this increased complexity, which would eventually become unmanageable, a handful of leading *Original Equipment Manufacturers* (OEM) and Tier 1 suppliers from the automotive industry, decided to work together to form a new standard that would serve as a platform for future vehicle applications, and their result is AUTOSAR[2].

The development of AUTOSAR is today a global cooperation between car manufacturers, suppliers and other companies from electronics, semiconductors and software industry. The cooperation started in 2003 with just a couple of the larger companies from the automotive industry and together they been developing a standardized software architecture for the automotive industry that is still being evolved. Together they have come up with their motto [2]:

*"Cooperate on standards, compete on implementation"*

This summarizes in a good way what their intention is. One important thing to remember is that AUTOSAR does not specify a specific implementation; it is up to the tool vendors and other companies to follow the AUTOSAR standard specifications and implement according to them.

### 2.2 Purpose with AUTOSAR

The development is moving from a hardware and component-driven process towards a process that is requirement and function-driven. The aim is not to optimize every single component, the future challenge for engineers is to optimize on a system level. In order to do so, a common platform is required that is scalable and supports exchangeable software modules. One of the fundamental ideas behind AUTOSAR is reusable Software Components (SWCs) that can deal with the increasing complexity today and in the future [2].

As mentioned earlier, software is tightly coupled with the ECU where it is going to be executed. If something is changed in the ECU, the software must be rewritten to suit with the hardware. It is problematic to buy software from one manufacturer and hardware from another, if they are not made to work with each other. Another example is for example one car manufacturer that has done all the work for one car in one of their production series, has to redo much work again to make the first system fit another car in a different production series. Depending on the manufacturer, models used and how variations are handled, this problem can be smaller or larger but the problem is still present in different scales.

The left side of Figure 3 shows the conventional integration between hardware and software. The figure represents the software tightly coupled for a specific hardware solution. Possible hardware

options could for example be changed architecture of the processor or different types of peripherals. These types of specific solutions might be better for certain situations, but with the increasing complexity it will probably fail in the long run. What the automotive industry wants is to move from the left side version and end up at the right side version instead. The applications are decoupled from the hardware (hardware independent) thanks to standardized interfaces that AUTOSAR can offer. With a standardized interface it would be possible to buy software and hardware from different manufacturers, and they would all work together. This would not be as smooth with the conventional solution.

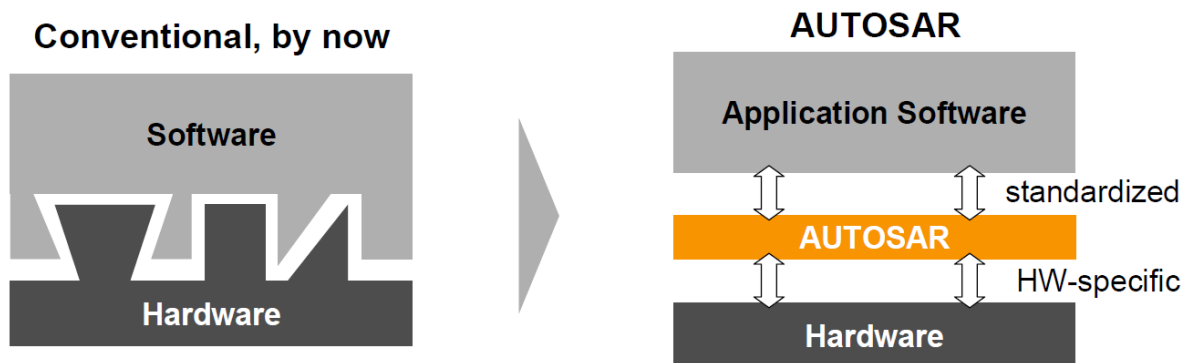


Figure 3: AUTOSAR makes the application software independent from the hardware

Application software that supports the AUTOSAR standard will receive several benefits. The following examples are the driving forces why we need AUTOSAR as listed in [3]:

- Manage increasing E/E **complexity** – Associated with growth in functional scope.
- Improve **flexibility** – More room for updates, upgrades and modifications.
- Improve **scalability** – The system can in a more graceful manner be enlarged.
- Improve **quality and reliability** – Proven software applications can be reused.
- Detection of errors in **early design phases**

***This answers Q1.1 “What is the AUTOSAR standard and why is it created?”***

AUTOSAR is standardized software architecture developed in cooperation between car manufacturers originally intended for the automotive industry but is steadily gaining interest from other industries as well. AUTOSAR was developed with the intention of being able to handle the increased complexity in today’s automotive industry and to decouple software from hardware.

### 2.2.1 Technical goals and objectives

Some of the primary goals for the AUTOSAR development may have already been mentioned, but this section will in short present the goals with AUTOSAR. According to their own website and the FAQs section[2], the primary goal is the standardization of basic system functions and functional interfaces, the capability to integrate exchange and relocate functions within a car network and substantially improve software updates/upgrades over the vehicle lifetime.

Some of these goals (mentioned below), will be tested in our demonstrator system to see to what extent they are fulfilled in practice with supported AUTOSAR tools.

The goals mentioned in the above can be extracted and pinpointed down to these points:

- **Modularity** - This will enable *“tailoring of software according to the individual requirements of electronic control units and their tasks.”* [4]. This goal is achieved by introducing standardized interfaces and hardware abstraction.
- **Scalability** – This will ensure *“the adaptability of common software modules to different vehicle platforms to prohibit proliferation of software with similar functionality.”* [4]. This goal can be achieved thanks to a combination of the other goals and the use of a standardized framework.
- **Transferability** - Transferred functions will *“optimize the use of resources available throughout a vehicle’s electronic architecture.”* [4]. This is achieved by hardware abstraction; the software should be possible to place on any ECU in the system.
- **Re-usability** – Reuse of functions will *“help to improve product quality and reliability and to reinforce corporate brand image across product lines.”* [4] This goal is achievable thanks to the object oriented system design. The software is designed as atomic components which should increase the reuse possibilities.

Similar and additional goals and objectives are these ones, as listed in [5]:

- Implementation and standardization **of basic system functions** as an OEM wide “Standard Core” solution.
- **Integration** of functional modules from **multiple suppliers**
- **Maintainability** throughout the whole “Product Life Cycle”
- Increased use of “**Commercial off the shelf hardware**”
- **Software updates** and upgrades over vehicle lifetime
- Consideration of availability and **safety** requirements
- **Redundancy** activation

### 2.2.2 Benefits suggested by the AUTOSAR Consortium

The AUTOSAR standard has many benefits to offer and several of them are stated on the official website for the Consortium. A summary of benefits are listed in Table 1 as listen in [2]:

General benefits	<ul style="list-style-type: none"><li>• Increased re-use of software</li><li>• Increased design flexibility</li><li>• Clear design rules for integration</li><li>• Reduction of costs for software development and service in the long term</li><li>• OEM overlapping reuse of non-competitive software modules</li><li>• Focus on protected, innovative and competitive functions</li></ul>
Specific benefits for OEMs	<ul style="list-style-type: none"><li>• Functions of competitive nature can be developed separately</li><li>• Later sharing of innovations is accessible</li><li>• Standardized conformance process</li></ul>
Specific benefits for suppliers	<ul style="list-style-type: none"><li>• Reduction of version proliferation</li><li>• Development sharing among suppliers</li><li>• Increase of efficiency in functional development</li><li>• New business models</li><li>• Preparation for upcoming increase in software volume</li></ul>
Specific benefits for tool developers	<ul style="list-style-type: none"><li>• Common interfaces with development processes</li><li>• Seamless, manageable, task optimized (time dependent) tool landscape</li></ul>
Specific benefits for new market entrants	<ul style="list-style-type: none"><li>• Transparent and defined interfaces enable new business models</li><li>• Clear contractual task allocation and outsourcing of software-implementation accessible</li></ul>

Table 1: AUTOSAR Benefits

***This answers Q3.1 "Why migrate to the AUTOSAR standard?" and Q3.8 "What are the benefits with AUTOSAR compared to non-standard solutions?"***

If a system has an unnecessary high software-hardware dependency that is possible to avoid, had a different approach been considered, AUTOSAR could very well be the solution. In most cases benefits such as reusability between vendors and scalability outweighs the trade-offs which could be introduced but each system is different thus there is no one-answer-fits-all scenario.

### 2.2.3 Main topics of AUTOSAR

AUTOSAR is more than just a new standard how applications should be written. In order to fully handle the problems and offer a complete solution, the project is centered on three main topics, see Figure 4.

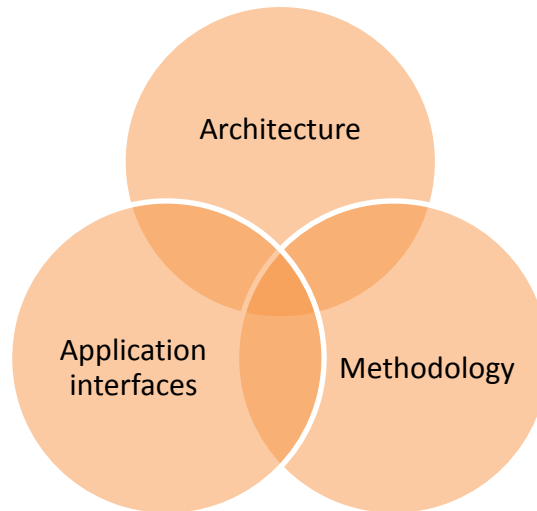


Figure 4: AUTOSAR main topics

These topics are first briefly described in the next three following sections, then as separate chapters (Architecture in section 2.4 , Methodology in chapter 2.6 and Application Interfaces which has no own section).

#### 2.2.3.1 Architecture

The first topic handles the layered software architecture. It includes a complete stack of the Basic Software (BSW) and how the relationship looks like between these layers. In addition to the BSW layer, the AUTOSAR architecture also distinguishes (on the highest abstraction level) the Runtime Environment (RTE) and Application Layer. The BSW layer is divided into smaller layers; these will be presented in detail in section 2.4.1.

The reader who is interested about the layered software architecture, we recommend the following document [6].

#### 2.2.3.2 Methodology

In order to exchange and combine SWCs from different manufacturers, a common methodology is necessary for some of the steps in the system development. It describes how the description templates should look like or other common formats that make a seamless configuration possible, between the different stages in an AUTOSAR development phase. So this topic in short manages the common formats that must be followed for a seamless design between different parts.

The methodology doesn't define a precise order how every single step should be taken, it defines dependencies between work and result e.g. what artifacts is needed to finish activities.

To read more about this, we recommend the following document [7].

### 2.2.3.3 Application interfaces

With the new release of AUTOSAR 4.0, the scope of standard is the following five domains [8]:

- Body and Comfort
- Powertrain
- Chassis
- Occupant and Pedestrian Safety
- HMI, Multimedia And Telematics

For each domain, there are a set of Standardized Application Interfaces that should be used to share content between SWCs. AUTOSAR does not specify the internal design of software applications; this opens the competitive window for developers. With a Standardized Application Interface, the exchange of data is clarified.

With common interfaces for specific domains, it is possible to exchange applications between different OEMs, suppliers etc. Thus, a cornerstone for reusability.

An example of a Standardized Application Interfaces for a Wiper/Washer component in the Body and Comfort Domain [9] can be seen in Table 2.

	Component: Washer	Component: Wiper
	WshngReq1 (WashingRequest)	WipgAutReq1 (WipingAutomaticRequest)
<b>Interfaces</b>	WshrSts1 (WasherStatus)	WiprSts1 (WiperStatus)
	WshrFldSts1 (WasherFluidStatus)	WinWipgCmd1 (WindowWipingCommand)

Table 2: Interfaces for Washer and Wiper components

These six interfaces are defined for the Body and Comfort domain, but you are however free to define your very own interfaces if you are creating a system that does not use any of these common functionalities or any widely required interfaces. They must however follow the standardized naming convention along with other rules, but that is something that the tools should take care of.

For further reading, we recommend [8] and the documents found at [10].

## 2.3 Common applications using AUTOSAR

There are number of different applications or systems that could be realized using the AUTOSAR standard. We read the latest version (release 4.0) of the domain documents and summarized some of them (Table 3) which can be found in the documents for the different domains that are specified in the standard [11][12][13][14][15].

Access subsystem	Visibility subsystem	Comfort subsystem
Anti Theft Warning System	Exterior lights	Seast Adjustment
Central Locking	Interior lights	Seat Climatization
Passive Entry	Wiper & washer	Sunroof/Convertible control
Immobilizer	Defrost Control	Steering column adjustment
Remote Keyless Entry	Mirror Adjustment	
Keypad	Body Sensor	
	Mirror Tinting	

Table 3: Summary of common applications

As stated, this is just a sample of all the applications or systems that could be found in the documents, but hopefully it gives a hint of what could be realized in AUTOSAR. There are no limitations for the behavior of the applications. An exception is applications that have requirements that might not be feasible to reach when following the AUTOSAR standard. These requirements could for example be timing, memory-usage etc.

***This answers Q1.6 “What are typical AUTOSAR applications?”***

Frequently used car functionality is suitable for the AUTOSAR standard. The scope of the standard is not limited to only the automotive industry since the implementation of a SWC’s behavior is not restricted.

## 2.4 AUTOSAR architecture

The AUTOSAR architecture is using a layered approach consisting of a total of three software layers running on top of a microcontroller (see Figure 5) [6]. These three layers are called, starting at the bottom, BSW, RTE and finally Application Layer (AL). An introduction to each layer will be presented below.

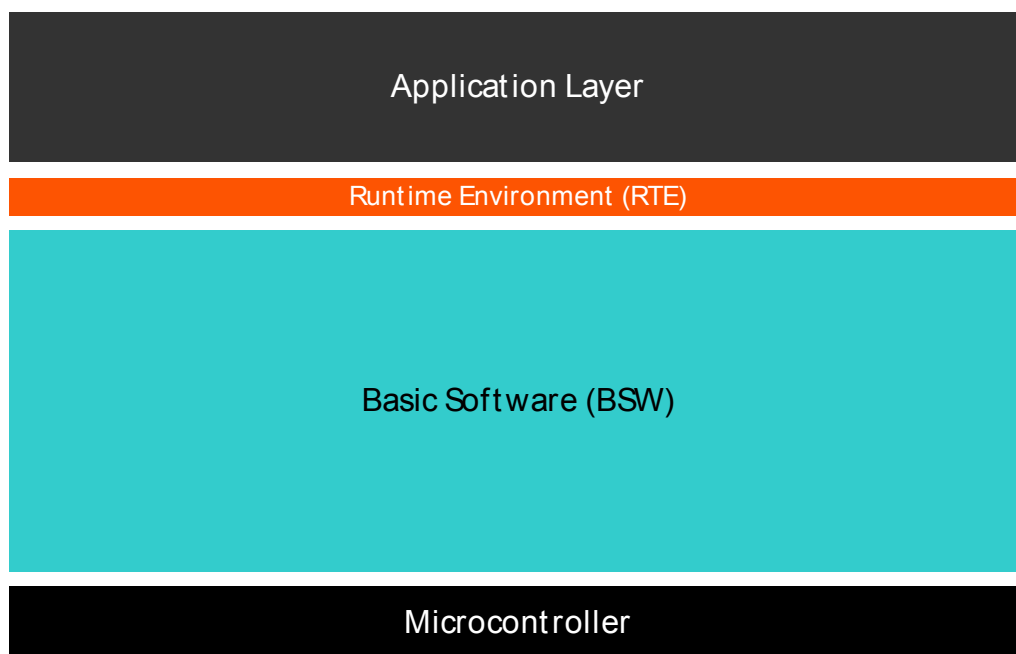


Figure 5: The AUTOSAR architecture.

### 2.4.1 Basic Software

While the BSW layer is a layer in itself, internally it consists of four sub-layers (see Figure 6). These sub-layers can be divided into functional groups (stacks), such as memory and I/O, vertically spanning the entire BSW layer.

#### 2.4.1.1 Microcontroller Abstraction Layer

The Microcontroller Abstraction Layer (MCAL) is located at the very bottom of the BSW layer. MCAL uses its internal software drivers to directly communicate with the microcontroller. These drivers



include: memory-, communication- and I/O drivers. The task of the layer is to make layers above it microcontroller independent.

When the MCAL is implemented it is microcontroller dependent but provides a standardized- and microcontroller independent interface upwards in the stack thus fulfilling its purpose.

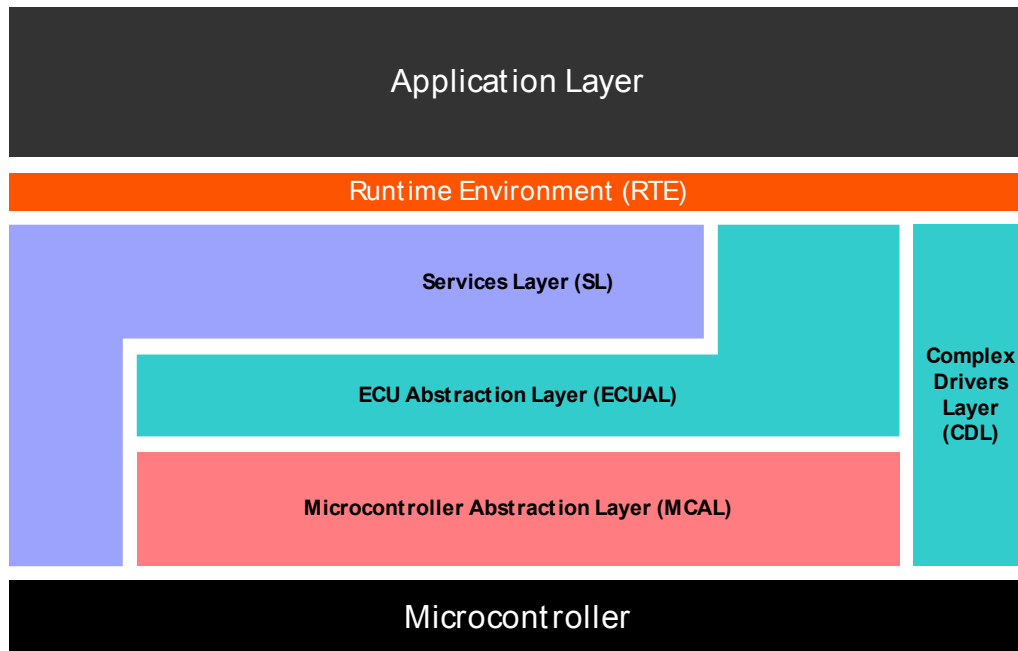


Figure 6: The AUTOSAR architecture including BSW sub-layers.

#### 2.4.1.2 ECU Abstraction Layer

Located on top of the MCAL is the ECU Abstraction Layer (ECUAL). Internally it has drivers for external devices and uses the interface defined by the MCAL to access drivers at the lowest level. Layers higher up in the stack can use the provided API to gain access to devices and peripherals without having to know anything about the hardware, for example, whether or not a device is located internally or externally, what the microcontroller interface looks like etc. All this is part of the ECUAL's task i.e. to make upper layers independent of how the ECU is structured.

During implementation the ECUAL is microcontroller independent thanks to the MCAL but dependent on the ECU hardware; layers interfacing the ECUAL are no longer dependent on either of them.

#### 2.4.1.3 Complex Drivers Layer

The Complex Drivers Layer (CDL) is one of two layers spanning the entire BSW layer. Typically this layer is used to integrate special purpose functionality and functionality currently being migrated from a previous system. Since this is the only layer between the microcontroller and the RTE, drivers for devices with strict timing constraints can benefit from being placed in the CDL as the multi-layered parts of the BSW layer is likely to introduce overhead due to additional layers and standardization [16]. Drivers for devices not compliant with AUTOSAR can also be put here.

Whether or not application, microcontroller or ECU hardware dependencies exist while implementing and afterwards for upper layers interfacing the CDL layer depends on the functionality one is going to integrate.

***This answers Q1.8 “How does the standard support migration from existing solutions?”***

The introduction and creation of Complex Drivers in the AUTOSAR standard can be used to migrate existing solutions.

#### **2.4.1.4 Services Layer**

The top sub-layer in the BSW layer is called Services Layer (SL). Along with operating system functionality the SL provides a collection of managers such as: memory-, network-, and ECU state management. This is also where diagnostic services reside.

As parts of the SL span the entire BSW layer it is not entirely microcontroller and ECU hardware dependent during implementation; its interface towards the RTE is however.

#### **2.4.2 Runtime Environment**

The RTE is the layer between the application layer and the BSW layer. It provides the application software with services from the service layer. All communication between SWCs, either on the same ECU or different ones, or services are done via the RTE. The main task of the RTE is to make the layer above and below it completely independent of each other. In other words, SWCs running on an ECU have no idea what the ECU looks like hence a SWC will be able to run on different looking ECUs without any modifications.

Logically the RTE can be seen as two sub-parts realizing different SWC functionality [17]:

- Communication
- Scheduling

When the RTE is implemented it is ECU and application dependent; thus it is specifically generated for each different looking ECU. Instead of adapting SWCs to different ECUs that is what is done with the RTE, this way SWCs can stay the same thus accomplishing the set out task.

#### **2.4.3 Application Layer**

The AL differs in two major aspects from the rest of the layers; it is component based and not standardized [8]. SWCs are such a key part of this layer and AUTOSAR and have as a result been awarded a dedicated section (see Software components). Looking at the RTE and the BSW layers there are lengthy standards documents specifying how they shall look and behave. Creating SWCs and the behavior in the AL can be done freely according to what a particular vendor wants. One restriction though is that all communication with other components, whether it is intra- or inter-communication, has to be achieved in a standardized way by using the RTE.

Since the RTE makes layers above it independent of hardware such as the ECU and the microcontroller the AL is, apart from a few cases, only dependent on the RTE [6]. For example the sensor-actuator SWC is dependent on the hardware but SWCs communicating with it are not.

***This answers to Q1.9 “How does the standard support reusability?” and Q1.10 “Is it possible to reuse software components?”***

A consequence of the hardware independence in the *Application Layer* is improved reusability of SWCs. This is done by using a standardized architecture.

## 2.5 Software components

AUTOSAR uses its own notation for modeling applications. An application consists of one or more SWCs based in the Application layer. In order for SWCs to communicate with each other they use the virtual functional bus (VFB). From a SWC’s point of view all it sees is the VFB and not the hardware dependent BSW and the hardware itself; in reality this is provided by the RTE [18]. The VFB as a whole is implemented not only by the RTE but also by the BSW. Whether or not two interacting SWCs reside on the same ECU is not important, both are connected to the same VFB and don’t need to know where in fact the other one is located [19].

Automotive functionality can either be fitted into a single component or spread out in multiple components. Apart from the actual implementation of a component it is always accompanied with a formal SWC description. A SWC description includes among other things:

- General characteristics
  - Name
  - Manufacturer
- Communication properties
  - PPorts
  - RPorts
  - Interfaces
- Inner structure (composition)
  - Sub-components
  - Connections
- Required hardware resources
  - Processing time
  - Scheduling
  - Memory

Having knowledge about hardware requirements for each SWC makes it easier to decide on what ECU a particular component should be placed. If you have the scenario where two manufacturers provide components of similar functionality but the slightly better one has a higher hardware resource demand, which would force you to add another ECU to your system, you would be able to make a decision at an early point on what to do. If you don’t have the ability to increase the number of ECUs in your system you are forced to go with the slightly poorer but less resource demanding component.

The center of attention when developing systems at the VFB level is its basic building blocks namely SWCs. A component has ports for which it uses to communicate with other components. Each port can only be assigned to one specific component. The complexity of the functionality implemented by a component can vary greatly so can the number of ports. AUTOSAR allows multiple instances of the

same component to exist; the same implementation but different memory spaces in order to store instance specific data.

### **2.5.1 Ports**

There are two types of ports in AUTOSAR SWCs; a PPort provides data defined in the port-interface while an RPort instead requests data. For sender-receiver and client-server port-interfaces AUTOSAR service versions also exist.

### **2.5.2 Port-interfaces**

AUTOSAR supports three types of port-interfaces: client-server, sender-receiver and calibration.

#### **2.5.2.1 Client-server**

The client-server pattern is well-known and commonly used. The server provides services while one or more clients may use its services to carry out tasks [20]. The client-server model defined in AUTOSAR is a simple  $n:1$  ( $n \geq 0$  clients, 1 server) one. An operation invoked on a server uses arguments supplied by a client to perform the requested task, these argument types can either be of simple (bool) or more complex nature (array).

When a client wants to invoke an operation of a server's interface a value for each operator parameter has to be provided. At a later point in time when a response is received it can either be a good or a bad one. There are a total of three possible responses:

- Valid response, the server was able to execute the requested operation and values have been assigned to all parameters provided by the operation interface.
- Infrastructural-error, a transmission to/from the server went wrong due to a broken bus [21]. For example an operation response never returned to the client resulting in a timeout on the client-side.
- Application-error, something went wrong on the server-side while executing the operation invoked by a client.

As mentioned above only an  $n:1$  client-server mechanism is supported by AUTOSAR. In more detail this means that any component acting as a client has to have an RPort connected to exactly one of a server's PPorts; this PPort can however be connected to an arbitrary number of client RPorts. Then it is up to the implementation of the communication to make sure every response is transmitted to the correct client's RPort. Since a component is not limited to one port or one port type, one and the same component can act as both a client and a server.

Limits exist on how clients may invoke operations on servers. Two concurrent invocations of the same operation on the same client RPort is not allowed; not until a response on the first invocation has been received, a good or a bad one, is it allowed to invoke the same operator again. Concurrent invocations of different operations are allowed on the same RPort but the VFB does not guarantee the ordering of the invocations i.e. the order in which the server sees the invocations or which order the responses from the server are received. The VFB is however required to enable a client to relate a response to a previous invocation.

When Components are modeled the client-server interface has its own graphical representation (see Table 4).





Port type	Service port	Icon
PPort	No	
RPort	No	
PPort	Yes	
RPort	Yes	

Table 4: Graphical representation of client-server ports.

### 2.5.2.2 Sender-receiver

This way of communicating allows one receiver to get information from multiple senders or a sender to send information to an arbitrary number of receivers. What type of data-element is sent between a sender and a receiver can be just about anything, from a simple integer to a more complex one like an array for example. There are two ways for which data-elements can be provided by the sender; either “last-is-best” which means the last value made available by the sender is the current one while “queued” means values are stored in a queue of predefined size.





A sender is completely decoupled from any receivers, it has no idea how many (if any) receivers are using the values it is producing. It is however possible at configuration time to enable transmission acknowledgements if a sender wants to know if a transmission succeeded or not.

If the “queued” semantics is used the receiver consumes values from the queue in a FIFO-fashion. The queue is located on the receiver side of the connection, this is logical since consumable reads are taking place. If a queue is full and there are still new values to add a queue overflow will occur and the new values are discarded; whether or not the receiver is notified of these events can be set at configuration time.

Considering multiplicity of the sender-receiver port-interface there are different limitations depending on which semantics is used. In the case of “last-is-best”, only 1:n (1 sender,  $n \geq 0$  receivers) communication is possible. Whereas for the “queued” semantics both 1:n and n:1 ( $n \geq 0$  senders, 1 receiver) are possible. The latter is possible because multiple senders can add values to the queue of a single receiver. For “last-is-best” there is only one value and if there are multiple senders they would constantly overwrite each other’s values and there is no guarantee the receiver will be able to see all produced values.

The VFB supports filters; filters can be used to reject values not fulfilling the conditions of a filter. If a value is filtered out, the current value is not changed, in case of “last-is-best” or “queued”, it is not added to the queue.

When modeling SWCs the sender-receiver port interface is differentiated from the client-server interface by its own graphical notation (see Table 5).

Port type	Service port	Icon
PPort	No	
RPort	No	
PPort	Yes	
RPort	Yes	

**Table 5: Graphical representation of sender-receiver ports.**

#### 2.5.2.2.1 Communication modes


Data reads and writes can be performed in two different ways, implicit or explicit. An implicit read means that a copy of the data is fetched before the start of the runnable (see Runnable entities) execution and is guaranteed to remain the same until the runnable terminates. Similarly for an implicit write data can be modified while a runnable is running and is not made available on the provider port for other SWCs to read until the runnable has terminated. In other words no explicit calls are made. On the contrary explicit reads and writes make API calls when they need to carry out the respective tasks; these API calls can be either blocking or non-blocking depending on the port- and runnable configurations.

#### 2.5.2.3 Calibration

Port-based calibration requires a component in need of calibration abilities to define an RPort. The RPort is connected via a connector to a PPort on a calibration parameter component; this is called public calibration parameters since the parameters are visible on the VFB. The PPort of a calibration parameter component can provide multiple SWCs with the same set of parameter values. But the case where components have their own calibration parameter component also exists.

SWC calibration can also be done in private internally within a composition; thus called private calibration. A sub-component is also known as a prototype and there are two types of calibration prototypes; either shared between other prototypes or individually assigned to prototypes in need of their own calibration parameter values.

As in the case for client-server and sender-receiver port interfaces calibration port interfaces have their own graphical notation as well (see Table 6).

Port type	Service port	Icon
PPort	No	
RPort	No	

**Table 6: Graphical representation of calibration ports.**

### **2.5.3 Connectors**

In order to connect two ports a so called assembly-connector is used. A connector can be used to both connect prototypes within a composed component as well as connect different component to each other.

### **2.5.4 Composed and atomic components**

A SWC can be built up by multiple sub-components and connectors. As mentioned earlier, in AUTOSAR these sub-components are also called prototypes. This type of component is defined as a composition of interconnected prototypes. A composed component is like any other component from the outside and it can be used to create hierarchical systems. An atomic component on the other hand cannot be divided into smaller components.

### **2.5.5 Component types**

In AUTOSAR there are seven types of components; a brief overview is presented below.

#### ***2.5.5.1 Application software component***

An application SWC implements either an entire application or just a part of one. The component is atomic and has access to all AUTOSAR communication types and services. The sensor-actuator component, which will be described shortly, is used to handle all interaction with sensors and actuators.

#### ***2.5.5.2 Sensor-actuator software component***

All sensor and actuator related tasks are handled by the atomic sensor-actuator SWC. It is sensor and actuator specific and makes it possible for other SWCs to use a sensor or actuator by providing an interface independent of the actual hardware. In order to do this it needs access to the ECUAL interface.

#### ***2.5.5.3 Calibration parameter component***

Its only task is to provide values for calibrating all connected components. If all connected components are not located on the same ECU the calibration parameter component has to be duplicated and placed on every ECU having at least one other component needing it. A calibration parameter component does not have any internal behavior as oppose to “normal” SWC.

#### ***2.5.5.4 Composition***

The composition type has already been described; one thing that can be added though is that it can use all AUTOSAR communication types and services.

#### ***2.5.5.5 Service component***

A service component in AUTOSAR is standardized and uses standardized interfaces to provide services. Direct access to BSW is needed in order to provide these services to other components.

#### ***2.5.5.6 ECU-abstraction component***

When the sensor-actuator component was described above it was mentioned that it interacted with the ECUAL. To be more precise, it is provided access to I/O via a client-server PPort on an ECU-abstraction component. This ECU-abstraction component has the ability to interact with other BSW.

#### ***2.5.5.7 Complex device driver component***

The final component type is the complex device driver component and it is a generalization of the ECU-abstraction component. It can communicate directly with other BSW and is typically used for

applications with critical resource demands. Software residing here is not as tightly coupled to the AUTOSAR standard as everything else but the interface towards the so-called “AUTOSAR world” does have to follow the AUTOSAR port and interface specifications [6].

### 2.5.6 Runnable entities

A runnable entity (runnable in short) is the smallest code fragment inside a SWC. It is these runnables that are mapped to OS tasks and will execute the behavior of the SWC. This is needed since the SWCs themselves are not aware of the BSW layer where all the OS functionality such as tasks resides. A SWC can have multiple runnables which may be required to carry out certain tasks.

Two main categories of runnables exist, the determining factors of which category a runnable is placed is described below.

- **Category 1** is where all runnables without any waiting points (non-blocking) are placed; in other words the runnables can with certainty be considered to terminate in finite time. Within category 1 there are two sub-categories; **1A** which is only allowed to use an implicitly defined APIs and **1B** which is an extension to 1A allowing it to also use explicit APIs and make use of functionality provided by a server.
- **Category 2** comprises runnables with at least one waiting point. With a few exceptions all runnables of category 2 strictly have to be mapped to an extended task since it is the only task type providing the *waiting* state [9].

Events of the RTE simply called RTEEvents (see Table 7) trigger the execution of the runnables; this is done by either activating them or waking them up.

Name	Communication restriction	Description
<b>TimingEvent</b>	None	Triggers a runnable periodically.
<b>DataReceiveEvent</b>	Sender-receiver only	Triggers a runnable when new data has arrived.
<b>OperationInvokedEvent</b>	Client-server only	Triggers a runnable when a client wants to use one of its services provided on a PPort.
<b>AsynchronousServerCallReturnsEvent</b>	Client-server only	Triggers a runnable when an asynchronous call has returned.

Table 7: A subset of all the available RTEEvents defined in AUTOSAR; for a complete list look in [17].

#### 2.5.6.1 Mapping to OS tasks

Mapping runnables to OS tasks is not the easiest job. There are a lot of properties playing a part when determining which type of OS task is required for the runnable at hand. These are some properties to consider: placed is described below.

- Runnables using an implicit API for reading and writing data cannot have any waiting points or infinite loops. In other words, runnables of category 1.
- Runnables of category 1 can be mapped to either task type (basic and extended).
- As already mentioned a runnable of category 2 normally needs to be mapped to an extended task. For example a synchronous server call will block until the server has completed its



execution resulting in a waiting point thus a category 2 runnable. The following exceptions exist:

- If no timeout monitoring is needed a runnable making a synchronous call can be mapped to a basic task.
- A basic task can also be used if the runnable of the invoked server is of category 1 and is invoked directly.

A couple of example scenarios can be used to illustrate the importance of some of the properties mentioned above.

#### 2.5.6.1.1 Scenario 1

The following is given for this scenario:

- Category 1A runnable(s).
- All communication is done with implicit reads and writes of data using the sender- receiver interface.
- A TimingEvent is used to trigger the runnable.
- Mapping to one task only.

##### 2.5.6.1.1.1 Basic task

If only one runnable exists it can be mapped to a basic task without any additional conditions. However if more than one runnable is present all of them have to be assigned the same cycle time and the sequence for which they will execute. See Figure 7 for an illustration.

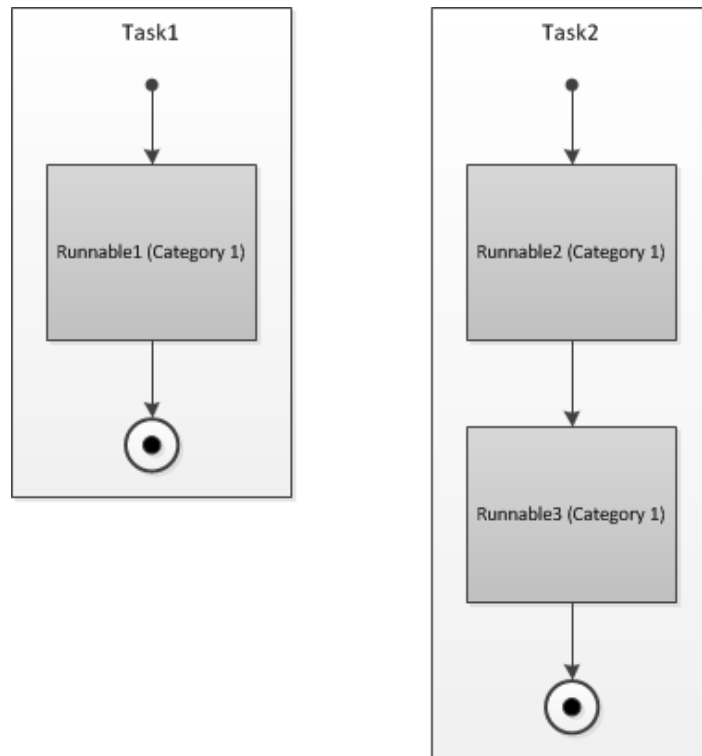


Figure 7: Example of scenario 1 when one (left) or two (right) runnables are mapped to basic tasks.

#### 2.5.6.1.1.2 Extended task

For the case in the basic task section where multiple runnables are used and no order of execution is provided an extended task will be used instead. This extended task will endlessly check for events such as OS alarms related to the different cycle times of the runnables in order to determine which runnable should be executing (see Figure 8).

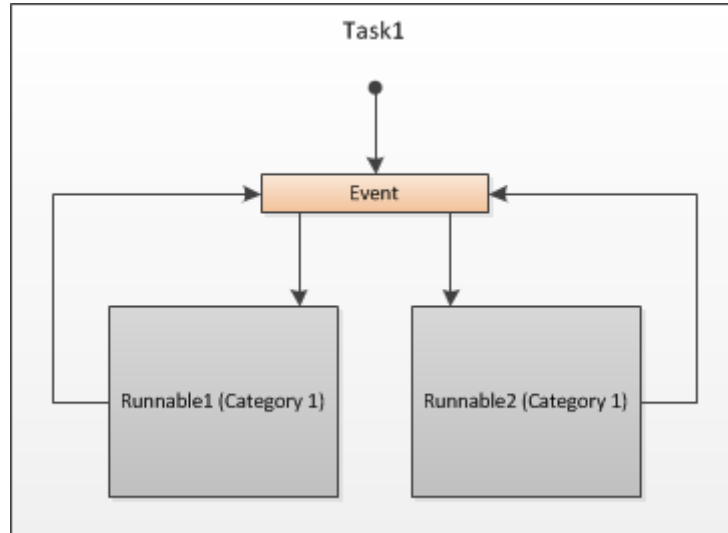


Figure 8: Example of scenario 1 when a mapping to an extended task is required.

#### 2.5.6.1.2 Scenario 2

The following is given for this scenario:

- Category 2 runnable(s).
- Explicit sender-receiver communication
- A DataReceivedEvent referenced by a waiting point.

When considering the normal case of this scenario the only possible task type to use is the extended task since a category 2 runnable is being handled. When an RTEEvent triggers the execution of the runnable it will eventually end up at the waiting point which will cause a block until data is received. Data reception is signaled by a DataReceivedEvent which will allow the runnable to continue execution.

Since there is no way of knowing for how long the runnable will remain in the blocking state each category 2 runnable is usually mapped to its own task (see Figure 9). Mapping two category 2 runnables to the same extended task could lead to serious time delays if one runnable comes out of blocking mode while the task is still waiting because the other runnable is blocked [17].

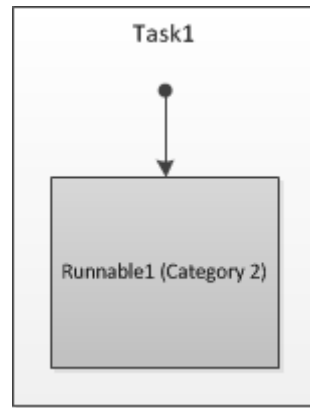


Figure 9: Example of a category 2 runnable mapped to its own extended task.

## 2.6 Methodology

In this section the AUTOSAR Methodology will be described in more detail.

### 2.6.1 Templates

Objects in AUTOSAR are described using different templates, which contains a collection of attributes that are necessary to describe a certain subject of AUTOSAR. Subjects could for example be ECUs, SWCs or the whole system. These templates are den specified using UML, which is based on the AUTOSAR UML Profile, described in [22].

***This answers Q1.11 “Is it possible to reuse configuration of the Basic Software Modules?”***

The use of templates should make it possible to share files between tools and users.

### 2.6.2 Methodology steps

The methodology also describes sets of activities for the development of a system. There are four major activities and steps (Figure 10) that must be performed to generate the *ECU Executable* which will run on the ECU. Each activity is divided into smaller activities and this thesis will not describe each sub-activity, only one will be presented in detail (Configure System). The methodology does not describe a complete process; it does not define how many times these steps must be repeated.

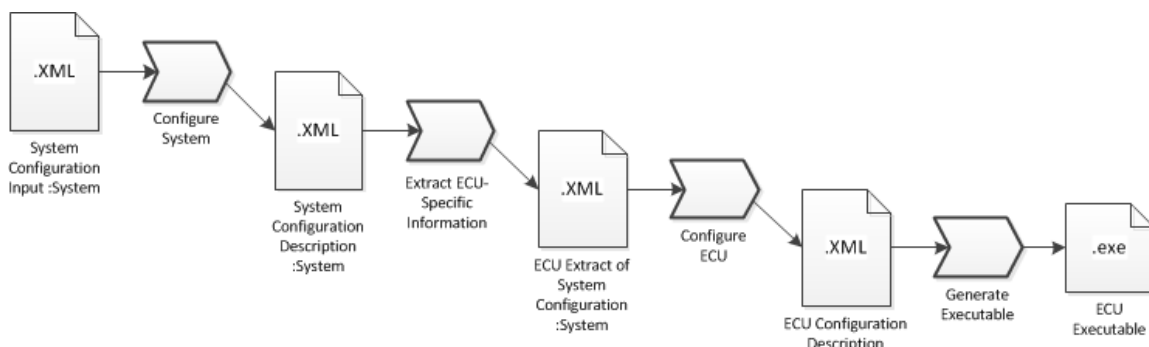


Figure 10: Main steps

### 2.6.3 SPEM

The AUTOSAR methodology is described with help from the Software Process Engineering meta-model (SPEM)[3], which is a standardized terminology used to describe processes. The standard is defined by the Object Management Group (OMG) consortium. Not the whole standard is used by the AUTOSAR methodology, just a small subset of SPEM is used and we will only present three of those modeling-elements. For a complete list and detailed description, we recommend [7][Methodology].

#### 2.6.3.1 Work-product



The document shaped figure represents a so called <<Work-Product>>. That is a piece of information that is either consumed or produced by an activity. There are four specific <<Work-Product>> defined in the AUTOSAR methodology [7]:

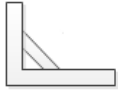
- XML-Document (may be one or several documents)
- c-Document (files containing sources written in C)
- Obj-Document (for object files)
- h-Document (files that are containing header files includes by c-files)

#### 2.6.3.2 Activity



The work-products just mention is consumed or produced by an <<Activity>>, which describes a piece of work that needs to be carried out. The name of the activity is written underneath the symbol. The role that is supposed to carry out the activity is not specified in the AUTOSAR methodology since it doesn't define any roles.

#### 2.6.3.3 Guidance



At last but not least, is the shape that is not exposed in the previous figure, the so called <<Guidance>>. In the AUTOSAR methodology, an <<Guidance>> is modeled to represent a tool that is needed to perform a certain activity.

### 2.6.4 Extract ECU specific information

We are only going to take a closer look at one of the activities shown in Figure 10 and that is the *Extract ECU specific information* activity. The rest of the activities are left unmentioned because the same work-flow is applied to those activities as well.

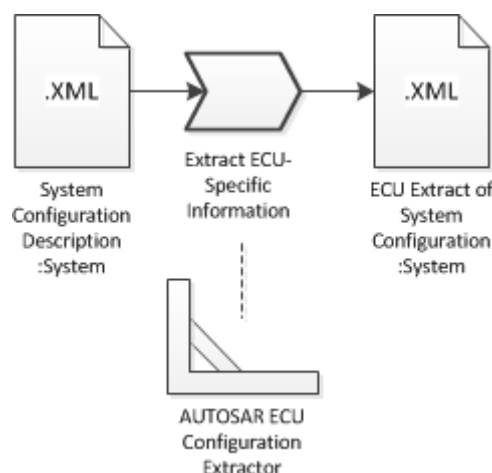


Figure 11: Configure system step in detail

What we see in Figure 11 is a more detailed view of the second activity. As we can see there is one input to this activity, namely *System Configuration Description*. In reality there are more dependencies with other documents describing the system and the SWC implementation, but we have chosen to not go into further details since [7] already has a great in-depth explanation for those that are interested.

One of the shapes in Figure 10 where not present in the previous figure and that is the <<Guidance>>, in this case called *AUTOSAR ECU Configuration Extractor*. Based on the input to the activity, the tool will produce the output.

In order to handle the activities, developers of AUTOSAR software need help from tools to grasp all the documents and dependencies. So as a part of our research, we have searched for vendors that offer AUTOSAR development or configuration tools. The ones we have found and examined are presented in section 0.

***This answers Q3.6 “Is it possible to integrate the AUTOSAR methodology into an already existing workflow?”***

Rather than defining a workflow, the methodology describes work packages and actions. The input which is required to perform a certain activity in order to generate a certain output is defined. This should make it possible to integrate the AUTOSAR methodology into existing ones.

## 2.7 Trade-offs

During the research and reading, numerous papers and articles have been encountered which propose possible benefits the AUTOSAR standard has to offer. Yet the AUTOSAR coin has two sides; benefits and trade-offs. Papers or articles that discuss the trade-offs that implementers of AUTOSAR compliant software and solutions must face is quite hard to find. There are a few papers that mention possible trade-offs and their effect [23], but almost none that is dedicated to just trade-offs. It might be the case that the standard is relatively new and therefore have not been exposed for in depth inspection by third party groups (researchers etc).

Since there are quite few articles written about the technical trade-offs that the AUTOSAR standard suffers from, it's hard to point at certain criteria.

We can at least expect that there should be trade-offs present when try to expose the standard for certain situations, which could indicate an ineffective system. Our intention is to test these plausible trade-offs, such as memory consumption, timing and execution overhead by exposing the system for certain tests. See section 7.4 for the evaluation done as a part of this thesis.

### 2.7.1 Memory

With the introduction of a complete architecture stack, there is likely to be an increase of memory use since an Operating System is needed and communication between SWCs always has to go via the RTE. This architecture stack is needed to decouple the software from the hardware, but the question to what price. We expect to see a larger increase of memory usage in simpler AUTOSAR solutions compared to non-AUTOSAR solutions. There might not be necessary to have a complete operating system in these cases, thus, non-AUTOSAR solutions are expected to be smaller.

### **2.7.2 Execution Overhead**

Since the software is not allowed to make direct calls to the hardware, there are likely to be an increase of the overhead in the system. You are not allowed to read a specific port directly; lower layers must handle this in order to decouple the software from the hardware. We are very confident that this will cause extra overhead compared to a “barebone” solution where direct calls to the hardware is allowed. The question is the magnitude of this overhead compared to the “barebone” solution. It should be possible to save some execution time by using the Complex Driver module in the BSW, since there is room to do non-AUTOSAR standard calls here.

### **2.7.3 Migration to AUTOSAR**

When introducing a new standard there are likely to be problems and child disease that could be considered as trade-offs. Companies may already have optimized procedures when a new system for the automotive industry is going to be developed. Introduce a new standardized way to develop systems into an already working system is prone to suffer from various problems.

The standard tries to comprehend this possible problem by presenting a methodology that should be followed, previously presented in section 2.6.

Developers and implementers need to understand the standard in order to create systems that are compliant with other AUTOSAR solutions. A complete understanding of the whole standard might not be necessary, but some of the fundamental parts, such as the architecture. The problems a migration could create could be interpreted as a kind of trade-off.

### 3 Dependability and Safety

This section will briefly present dependability, safety and at the end how AUTOSAR addresses safety issues.

#### 3.1 Critical systems

Users of commonly used desktop software know that there is a chance that the software can sometime fail. That is why for example we save this document several times while we are writing, and keep backups saved in different locations. A failure in the software can often be disturbing and hours of work can disappear in a wink of an eye, but in the long term, it doesn't lead to serious damage.

However, there are other types of systems where failures in the software doesn't just lead to inconvenience, it can lead to, for example: physical damage, huge economical loss or human lives at stake. These systems that people and business depend on are called *critical systems*. If these systems fail to deliver the expected service then the result may be serious problems and economic loss, as mentioned earlier.

In [20], three different main types of critical systems are mentioned:

1. *Safety-critical systems* – Failure may result in injury, loss of life or serious environmental damage.
2. *Mission-critical systems* – Some kind of goal-directed activity may fail. This could for example be a navigational system for a spacecraft.
3. *Business-critical systems* – Failure may result in very high economical cost for the business using the system. An accounting system in a bank is a good example.

Here we will only focus on safety-critical systems since this is strongly connected with the automotive industry, where AUTOSAR has its origin. Cars are for example dependent on safety-critical systems, such as airbag or brake systems.

#### 3.2 Dependability

The most important property of a critical system is its *dependability* and there are several reasons [20]:

1. *"Systems that are unreliable, unsafe or insecure are often rejected by their users"*.  
If the customers cannot trust the software, they will refuse to use it and may also reject other products from the same company.
2. *"System failure cost may be enormous"*.  
The cost of the control system is nothing compared to the cost of a system failure.
3. *"Untrustworthy systems may cause information loss"*.  
It is not just expensive to collect data, but also to maintain it. The data itself is often much more valuable than the system it is processed on. To guard the data against corruption, much effort and money is spent on e.g. duplication.

A high cost is always involved when talking about critical systems. Therefore, only well-known and proven development techniques and methods are typically used. It is better to stay conservative rather than embrace new methods that might be more effective, but without long-term testing.

### 3.2.1 Principles

There are four different principles related to dependability [20]:

1. Availability – The ability of the system to deliver services when requested.
2. Reliability – The ability of the system to deliver services as specified.
3. Safety – The ability of the system to operate without catastrophic failure.
4. Security – The ability of the system to protect itself against accidental or deliberate intrusion.

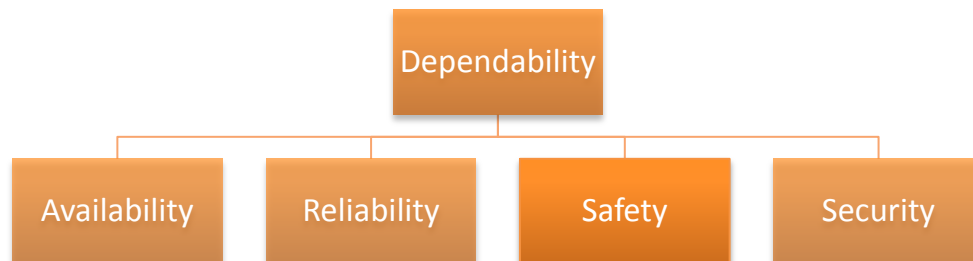


Figure 12: Dependability and its sub categories

One key part in safety is that no matter how good the implementation quality is, it cannot be considered to be safe if the specifications are incorrect. So it is very important that the specifications are formulated correctly, with formal notation or with words.

### 3.3 AUTOSAR and Safety

Previous releases of the AUTOSAR standard (R3.1 and older) had no explicit focus on safety and security [24]. With the latest release (R4.0) functional safety concepts has been introduced to support safety related applications. Some of these new concepts are [8][24]:

- Memory partitioning – Applications are separated from each other to avoid corruption of data.
- Support for dual microcontroller architecture – The aim is to detect faults in one core by a secondary unit.
- Safe end-to-end communication – Aims to provide the applications means to send data and detect errors during transmission.

Since the latest revision is so new, most development tools are still adapted for the R3.1 standard as seen in Table 9, Table 10 and Table 11.

#### 3.3.1 Standards

The introduction of these new concepts is however not enough to develop safe applications. Standards such as IEC-61508, ISO-15998 and ISO-26262 must be followed before the applications can be considered as sufficiently safe. ISO-26262 is an adoption of IEC-61508 for the automotive industry and defines different levels of Automotive Safety Integrity Levels (ASILs) [25]. Depending on the ASIL level set for the application to be developed, certain activities must be performed based on the ASIL level.

To use the generated code one of the following options are available in a safety related project:



- **Qualified** – Tools which generate code based on input must be qualified. It means that the tool and the generation of code have gone through extensive testing and evaluation, before the generator can be qualified.
- **Tool proven-in-use** – The latest versions of tools are never used. Instead, versions of the tools which have been used by many for a sufficiently long period of time and are well documented. The vendor also must have systematic procedures to handle errors reports and correct these or document the errors.
- **Review of generated code** – The generated code from the tools must go through the same rigorous review process as the newly developed code.

The AUTOSAR Methodology is designed to use tools for certain activities. Many of these tools generate code based on information found in the descriptions files for example SWCs. None of the tools we have examined are qualified and the two used during the thesis are not stable enough to be considered as proven-in-use. This is not so surprising since safety concepts have been recently introduced in R4.

***This answers Q1.12 “How does the standard address safety issues?”***

In the latest revision of the AUTOSAR standard (R4), concepts for functional safety have been introduced. The AUTOSAR standard is young and the tools are still under development.

## 4 Survey of Development tools for AUTOSAR

As mentioned earlier in section 2.6, AUTOSAR defines a methodology which defines the main development activities. In order to handle all these steps that are required, certain tools must be used.

### 4.1 Tool vendors

In this section we will present different tool vendors that we have examined.

#### 4.1.1 Research

AUTOSAR has a number of partners of different level. These partners focus on a bunch of well-defined areas including OEMs, semiconductor manufacturers, tools and services providers etc [5]; the latter is the interesting one for this section. Once we had located a collection of vendors [5] [26], Google was used to gather further information about each company and what services they provide. It was discovered that all companies did not supply development tools and thus were discarded from the list of candidates to be compared and eventually contacted at a later stage. A short presentation of each tool vendor is presented below.

##### 4.1.1.1 Elektrobit – Tresos

The Tresos family suite from Elektrobit (EB) aims at the configuration of RTE and BSW. It does not offer a complete tool-chain in the development of AUTOSAR compliant ECUs. The whole suite consists of 5 different parts and of those parts; there are 3 that are more interesting [27]:

- EB Tresos Designer – A design tool that is used to design the bus used in the system. It is possible to extract a *Communication Matrix* which is a part of the *System Configuration Input*.
- EB Tresos Studio – Eclipse based configuration, validation and generation tool for BSW. According to the vendor, it supports the AUTOSAR methodology.
- EB Tresos AutoCore – As we have mentioned before, BSW is the part in the AUTOSAR architecture that are dependent on the HW. AutoCore is EBs BSW core and consists of more than 40+ BSW modules, configured using Tresos Studio.

No simulation possibilities are mentioned on their website, which is not too surprising since simulation is about the SWC and that is not the aim for this vendor.

##### 4.1.1.2 Mecel – Picea

The Picea suite from Mecel provides a complete tool-chain for AUTOSAR software development, from SW-C down to the BSW. This suite consists of different tools and if we count the optional simulation software there are 4 tools in total [28]:

- *Picea Workbench* – Eclipse based tool for configuration of the RTE and BSW. Can also be used to edit the SW-C and generate the application layer.
- *Picea RTE* – Integrated into the Picea Workbench to handle the RTE generation. It is command line based and has the possibility to validate configurations files according to the AUTOSAR scheme.
- *Picea BSW* – AUTOSAR compliant BSW core together with a code generator. It is also command line based tool like Picea RTE. It has the possibility to generate C-code from

configuration files and validate these configuration files. How many modules that are implemented is not mention on the website.

- *Picea VFB Simulator* – This tool is created to increase the development efficiency of SW-C since it makes it possible to run SW-C without the actual HW. The AUTOSAR methodology does not specify any specific simulation tool.

#### **4.1.1.3 Mentor Graphics – Volcano**

The Volcano suite is made of 2 different tools that are used for AUTOSAR development [29]. Though it is quite cryptic to fully understand if they offer a complete tool-chain or not.

- *Volcano System Architect* – An Eclipse-based tool that is used to design the system. The mapping of SWCs to ECUs is supported by the tool (mapping is one part of the AUTOSAR methodology) and how the signals are exchanged between the components. It might be possible to develop the SWCs with this tool since it is Eclipse-based, but that is not a guarantee.
- *Volcano Systems Tester* – This tool is used to create a testing environment to test BSW modules on a PC. It supports the conformance test that is specified by AUTOSAR.

Note: Conformance tests are not specified before the AUTOSAR 4.0 release.

Our conclusion is that the Volcano suite is more of a support tool for AUTOSAR developers rather than a development tool. A development tool is according to us a tool or software that helps the developer implement solutions for a certain task.

#### **4.1.1.4 Infineon**

During our first research phase when we searched for AUTOSAR development tools this company ended up as a bookmarked item. Once we took a closer look what they had to offer, it turned out that they only offered MCAL modules, the modules in the BSW that are closest to the hardware[30]. It is possible to configure and use these modules with a tool from a different vendor, such as Vector or Elektrobits.

#### **4.1.1.5 Continental – CESSAR-CT**

This suite is a combination of RTE and BSW along with a configuration tool that is called CESSAR-CT. The tool is like many others based on Eclipse, but also the ARTOP framework. This should make it possible for the tool to handle the latest release of AUTOSAR. According to the vendor, the tool has a 100% AUTOSAR meta-model coverage. That means it is possible to travel through all the 4 main steps in the methodology earlier presented. The website does not give any straight answers if it is also possible to develop SW-C and how much help it will offer to the developer[31].

It might be the case that this vendor, just like Bosch, has other channels to promote their solution. We get the same feeling, if we buy their solution, they can offer AUTOSAR solutions and also have the tool to configure it according to our needs.

#### **4.1.1.6 Bosch – iSolar and Cubas**

The suite for developing AUTOSAR solutions from BOSCH consists of two parts[32]:

- *iSolar* – Just like many other tools, this is a Eclipse-based development software. The information on their website is unfortunately quite limited for unknown reasons. According to their website it is possible to configure all AUTOSAR layers with this tool and it supports

AUTOSAR release 2.0 and upwards. We are not sure to what extent it is possible to develop SW-C using iSolar, it is at least possible to configuration across all layers.

- Cubas – This is the package that contains the AUTOSAR BSW core modules. How many of the defined modules they have implemented and offer is unknown, back in 2008 most of the modules that handle memory, communication and services were finished. How it looks today is unknown.

When we examined the tools offered by Bosch we felt that we are not the customer they are looking for. With over 250 000 employees [33] it is a very large company and we are quite sure they have other channels where they promote their software and solutions. That could be the explanation why there is so little information about iSolar on their website.

#### **4.1.1.7 pulse-AR – Quasar**

This suite from pulse-AR only consists of one tool called Quasar. This tool is still under development but is going to be based on Eclipse and the ARTOP framework. That means it will support AUTOSAR release 4.0 once it is finished. The tool will offer a way to develop SW-C and design the system once it is released [34].

#### **4.1.1.8 dSpace – SystemDesk and TargetLink**

dSpace's development suite consists of SystemDesk and TargetLink. SystemDesk allows you to design network and software architectures by modeling them according to the methodology defined by AUTOSAR [35]. The RTE generation module, which is available as an add-on, can generate a complete RTE based on your ECU and SWC.

TargetLink generates production code from the graphical representation of your architecture modeled in SystemDesk or a third party graphical environment like MATLAB®/Simulink/Stateflow [36]. Also included in the suite is a simulation module which lets the user simulate either a single ECU system or a more complex system containing an arbitrary number of ECUs using a PC. No BSW support is provided; neither a complete set nor the possibility to develop your own ones using their tools. The suite conforms to AUTOSAR 2.1, 3.0 and 3.1.

#### **4.1.1.9 Vector – DaVinci**

Vector provides a complete suite of development tools called DaVinci, a comprehensive set of efficient and scalable BSW modules and an ECU RTE; the latter two are part of the MICROSAR product family. The included BSW modules cover the entire range of BSW defined in the AUTOSAR 3.x releases [37]. The DaVinci suite supplied by Vector consists of: DaVinci Configurator Pro, DaVinci Configurator Kit, DaVinci Developer and DaVinci Component Tester [38].

DaVinci Configurator Pro makes it possible to generate and configure BSW created by Vector or anyone else. Comprehensive validation of modules is performed in order to guarantee consistency between modules.

DaVinci Configurator Kit enables the user to extend DaVinci Configurator Pro to make it easier to create new BSW modules or integrate already existing ones.

DaVinci Developer is used when designing SWCs and configuring the MICROSAR RTE. The design is done in a fully graphical environment.

Last but not least DaVinci Component Tester is the simulation environment used to test your SWCs designed in DaVinci Developer, this is done by emulating the VFB and which makes it possible to use a regular PC instead of having to use the target hardware.

#### **4.1.1.10 Geensoft – AUTOSAR Builder**

The suite provided by Geensoft is called AUTOSAR Builder; it is based on Artop (section 4.2) and includes the following tools [39]: AUTOSAR Authoring Tool (AAT), Generic ECU Configuration Editor (GCE), RTE Generator (RTEG), AUTOSAR Simulation (ASIM) and AUTOSAR Re-targeting Tool (ART).

The AAT is a tool for creating SWC, ECU and System descriptions. Full support for validating all AUTOSAR standard descriptions is included as well as automatic error correction to help avoid inconsistencies. As a plug-in to the ATT, one can use the ECU extract (EEX) in order to extract a specific ECU description from a vehicle description.

The ART speeds up development significantly by generating ready-to-be-used C code from MATLAB/Simulink models. Two steps are involved when using the ART. First of all a supported model is imported which enables mapping of systems/subsystems and SWC/runnables and finally a SWC template and re-targeted C code is generated and ready to be used.

ASIM is a PC based simulator tool focusing on the VFB and ECU levels. On the VFB level you are able to simulate SWCs independently of any hardware. SWC ports and interfaces are validated and verified as well as scheduling, triggering and activation of runnables. Testing on resource consumption can also be performed. The ECU implementation services provided by the BSW is validated and verified at the ECU level.

Based on the SWC description rules the SCVT verifies that your SWC descriptions are valid. It also provides static analysis of the code.

GCE is used to create and configure BSW parameters. Everything the user does is strictly checked to comply with the AUTOSAR standard.

There is also a plug-in suite including: AAT, EEX, Software Component Conformance Validation Tool (SCVT), GCE and AUTOSAR RM.

#### **4.1.1.11 ETAS - ASCET**

ETAS' ASCET product family provides a wide area of tools to help development of an AUTOSAR solution.

ASCET-MD (Modeling & Design) makes it possible to create SWCs either graphically or by using a high level programming language like C. In addition the created components can be simulated and it is possible to import Simulink models [40].

ASCET-RP (Rapid Prototyping) allows testing of software functionality in an ECU environment. The software functions are run on an OSEK operating system while connected to the bus system of the rapid prototyping system [41].

ASCET-SE (Software Engineering) is a certifiable code generator generating MISRA compliant code. The code can be efficiently stored and is runtime-optimized [42].

ASCET-SCM (Software Configuration Management) is, as the name implies, an interface to the version and configuration management system. It integrated directly into the ASCET development tools [43].

ASCET-DIFF (Model Difference Explorer) is a tool to easily track down changes in the system by comparing the models. How the comparison is executed can be customized by the user. ASCET-DIFF can be run separately or together with ASCET-MD [44].

ASCET-MDV (Model Viewer) gives a graphical overview look of the models. With this tool it is easy to get a feel of the design and architecture of model. An advanced search function is available in order for the user to easily locate classes, data elements etc.

Outside the ASCET suite INTECRIO can be used for virtual prototyping by testing functionality on a Windows PC. It has support for ASCET models, MATLAB/Simulink models, C code modules and AUTOSAR SWCs. In order to use AUTOSAR SWCs it makes use of an AUTOSAR RTE.

RTA-OS can be used to provide a real-time operating system which can be used together with the AUTOSAR 3.0 standard. RTA-OS can easily be integrated with other ETAS tools, for example RTA-RTE is ETAS' tool to generate an AUTOSAR RTE.

#### **4.1.1.12 KPIT Cummins**

KPIT Cummins supplies a suite of BSW modules including communication and diagnostics modules and memory services to mention a few. By providing BSW modules it helps customers to speed up the migration to AUTOSAR [45].

Along with the BSW modules KPIT Cummins has developed a set of tools to simplify deployment. The ECU Configuration Editor is used to check if modules are correctly configured, if not, the user will be notified of such event. An RTE Generator is also available providing the same validation possibilities.

KPIT Cummins also offers MCAL services, these services are divided into three categories:

- Microcontroller Drivers: Watchdog
- Communication Drivers: CAN (Controller Area Network), SPI (Serial Peripheral Interface)
- I/O Drivers: PWM (Pulse-Width Modulation), DIO (Digital Input Output)

No support for SWC development exists, a third party vendor is needed for that.

#### **4.1.1.13 The MathWorks**

With the help of MathWorks' products (Simulink, Stateflow, Real-Time Workshop and Real-Time Workshop Embedded Coder) customers are able to generate SWCs which can be used together with AUTOSAR. Using a combination of Simulink and Real-Time Workshop Embedded Coder, AUTOSAR SWC descriptions can be imported and exported as well as generated into code. Simulink also has simulation and rapid prototyping capabilities [46].

The MathWorks does not provide an entire AUTOSAR tool chain. In order to get one they recommend third party vendors like Vector and KPIT Cummins. Both these vendors have BSW and RTE generators which is what MathWorks is missing.

#### **4.1.1.14 Arc Core – Arctic Studio**

All development is done via Arctic Studio, an Eclipse based IDE. In order to create and generate the different parts of a complete AUTOSAR solution four plug-ins are available: BSW Builder, RTE Builder, SWC Builder and Extract Builder.

BSW Builder includes everything needed to edit and generate BSW. A host of BSW modules are supported such as CAN, DIO and PWM among others. Validation support is built in which will point you in the right direction when an error is discovered. If more than one version of a BSW module exists they can be compared and merged if desired [47].

To get a complete AUTOSAR solution an RTE is needed. The RTE Builder generates an RTE based on the SWCs input coming from the ECU Extract. When generating an RTE one step consists of mapping runnables to tasks provided by the OS BSW module; this is supported in RTE Builder. As with all the other tools developed by Arctic Core validation capabilities are included [48].

SWC Builder allows the user to edit and generate SWCs. While developing the validation mechanism will pick up on missing elements and erroneous configurations. To limit the risk of errors the tool is implemented to only allow the user to make choices that makes sense in the current context [49].

Extract Builder is used to connect, either manually or automatically, all SWCs in the system. Extract Builder can find SWCs defined in multiple files which makes it a lot easier when more than one user is involved in a project. To get an overall feel of all the system including its SWCs and connectors the connections view can be used [50].

Tools provided by Arc Core are created to support components and modules created using tools by another vendor. For instance, the RTE Builder can make use of ECU extracts created by not only Extract Builder but also by another tool from a different vendor [48].

Unlike other vendor suites this one is open source, but for commercial usages there are a couple of benefits from buying a commercial license: product does not have to be released under the GPL, technical support provided by Arctic Core and legal protection [51].

#### **4.1.1.15 Open Synergy – COQOS**

Open Synergy offers a suite called COQOS consisting of the following tools: COQOS-tP and COQOS-tC.

COQOS-tP is used to develop, run and test SWCs directly on a Linux based computer [52]. The Eclipse and Artop based generators are capable of generating both an RTE and an adaption layer for a Linux OS exist; the latter means an AUTOSAR OS is not needed. COQOS-eCOM and COQOS-ePDU are used to connect a computer running Linux to a CAN bus. A built-in validator is able to report any inconsistencies needed to have a look at.

COQOS-tC is a tool to verify that SWCs and RTEs are AUTOSAR compliant [53]. In order to perform the analysis a description of the ECU configuration and an execution trace provided by COQOS-tP are required. Model consistencies and constraints are fully checked in order to meet the AUTOSAR requirements. Once the analysis is completed, a detailed report is produced making it easier to take necessary actions if problems are discovered.

All tools comply with the AUTOSAR 3.1 standard.

## 4.2 Artop

In this section, the Artop framework for Eclipse will be briefly presented.

### 4.2.1 Background & motivation

The purpose of Artop is to provide basic AUTOSAR functionality that every tool vendor would have had to implement anyway if they developed their own tools from scratch [54]. This has the advantage that every tool vendor using Artop as a base will not have to re-implement the basics required by AUTOSAR but can focus solely on extending their tools with additional functionality enabling more user-friendly and time-efficient tools. In addition, after each new AUTOSAR release the time spent upgrading the tools in order to comply with the latest release will be significantly reduced. This is one of Artop's main goals and another one is interoperability, e.g. making tools from different vendors compatible with each other and this is easily achievable since they have a common base, namely Artop.

With the aim of realizing this, a group of companies with strong interest in AUTOSAR formed the Artop User Group. The goal is not to provide a complete ready-to-be-used tool chain but, as previously mentioned, set up a platform from which vendors can use to develop commercial products. There are three ways of taking advantage of Artop:

- Design members, companies in this group make decisions on what to focus on and how Artop should be designed. These companies include Continental and BMW Car IT to mention two.
- Contributing members, these companies take part in extending Artop and maintain already existing parts.
- Adopters, companies using Artop to develop tools for end-users are placed in this group. The only way they contribute to Artop is by providing feedback which can be used to improve the Artop platform even further.

### 4.2.2 Architecture

Tool vendors using Artop as a base will typically have a four layered architecture (Figure 13); whereas only the two in the middle are a result of Artop.

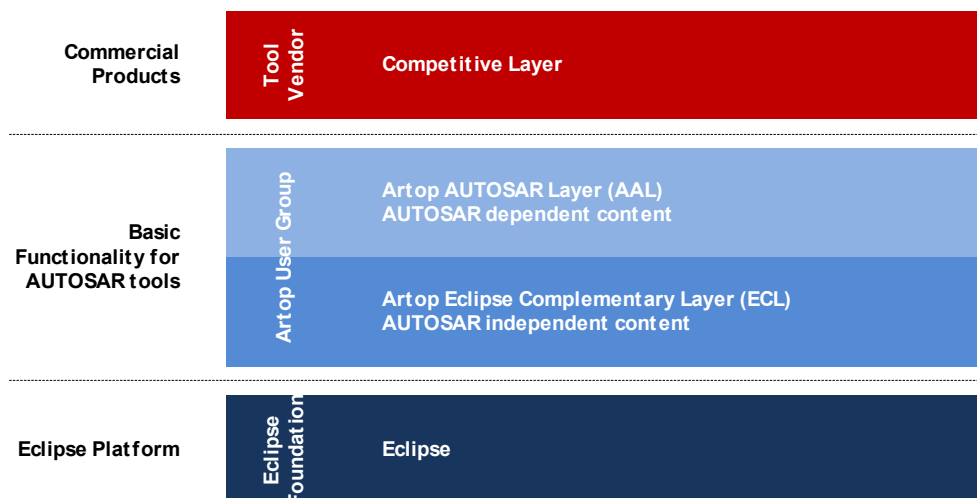


Figure 13: Artop Architecture



Since Artop is based on Eclipse the layer at the very bottom is the Eclipse platform. This allows an AUTOSAR tool to have useful development plug-ins such as version control which is typically found in any Eclipse based IDE.

The lower of the two Artop layers is called Eclipse Complementary Layer (ECL) and contains non-AUTOSAR specific parts only. The generic validation engine located on this layer can be used to minimize the task of validating AUTOSAR model constraints. Functionality for comparing AUTOSAR models is also included which allows for easy differentiation of AUTOSAR model versions and keep track of changes that have been made. Many Eclipse and Eclipse Modeling Framework extensions are also made available by the ECL. There are plans in the pipeline for upcoming versions of Artop to move the functionality of this layer directly to Eclipse which would allow the Artop user group to focus on the AUTOSAR specific parts.

Artop AUTOSAR Layer (AAL) is the other Artop layer and contains AUTOSAR specific functionality. The most important one is the AUTOSAR metamodel implementation. It supports all AUTOSAR releases starting from 2.0 and future releases will be integrated as well within the smallest possible timeframe after an official release. Among other features the AAL also provides serialization and de-serialization of XML based AUTOSAR models.

At the top of the layered architecture is the Competitive Layer; this is where all the vendor-specific functionality is located and their opportunity to stand out from the crowd. Any specific functionality requested or required by the end-user is placed in this layer. This is the only layer that does not look the same for different vendors.

## 5 Selection of Development Tools

In this section an iterative process will be presented in order to shorten the list of potential tool suites which will be evaluated and used when realizing our demonstrator. An initial list consisting of properties believed to be important to CrossControl will be used to compare all tool suites and eventually exclude the ones that come up short. If the first iteration completes with an insufficient number of tool suites, still in the running to be evaluated, a second iteration will commence with a new list comprising only a subset of the properties of the original one. If this scenario presents itself the important properties will be weighed against each other and the less important ones will be removed. Although there will only be time to evaluate two suites the goal is to end up with five or six in order to have a buffer in case some vendors do not respond or deny us the opportunity to evaluate their suite.

### 5.1 First iteration

To help differentiate the tool suites provided by the chosen vendors, an initial list of important properties has been compiled. The list is divided into categories and will be explained in detail below (see Table 8). Once the important properties have been introduced a side-by-side comparison will be made to easily see which properties the different tool suites support (see Table 9, Table 10 and Table 11).

Properties	Description
<ul style="list-style-type: none"> <li>• <b>Company</b> <ul style="list-style-type: none"> <li>○ <b>Name</b></li> <li>○ <b>Country</b></li> </ul> </li> </ul>	Company information like where in the world they are located can be of interest for CrossControl; a Swedish based company could be to prefer if for example meetings are to be arranged in the future, on-location support etc.
<ul style="list-style-type: none"> <li>• <b>Suite overview</b> <ul style="list-style-type: none"> <li>○ <b>Size</b></li> <li>○ <b>Supported AUTOSAR release(s)</b></li> <li>○ <b>License model</b></li> <li>○ <b>Price</b></li> </ul> </li> </ul>	For an overview, or quick look, of a suite there are a couple of interesting attributes. The amount of tools included, perhaps it is better to work with one tool rather than having to learn five for the same purpose. Or it simply gives an idea of how big the suite is, not necessarily in terms of functionality but in terms of size. Not all suites support the same AUTOSAR release, thus it is important to make sure the selected suite supports the release you are intending to use. The license model for both the suite and developed products are very important, tools enforcing developed products to be released as open source are not of interest for CrossControl. As always money is a factor, but this is more of interest to CrossControl in the future rather than for this thesis.
<ul style="list-style-type: none"> <li>• <b>Suite content</b> <ul style="list-style-type: none"> <li>○ <b>Development (layers and platform)</b></li> <li>○ <b>Simulation (layers and platform)</b></li> <li>○ <b>Included BSW</b></li> </ul> </li> </ul>	When it comes to the actual tools in the suites it is important to know how much of the tool chain they cover. Development of an AUTOSAR solution consists of three distinct layers, BSW, RTE and SWC. If the entire tool chain is covered by a single vendor it has the advantage of only having to deal with one vendor when support is needed. But if one vendor provide superior tools but not for the entire tool chain it should still be an option to use it together with tools from another

	<p>vendor. Platform support is another important aspect and should be taken into consideration; some ECUs may only have support for Linux or Windows XP embedded etc. A couple of vendors offer BSW as well to some degree, certain BSW modules are more interesting to CrossControl than others i.e. CAN communication modules and this should be looked at when comparing vendors who provide BSW.</p>
<ul style="list-style-type: none"> <li>• <b>Integrated development environment (IDE)</b> <ul style="list-style-type: none"> <li>○ <b>Eclipse based</b></li> <li>○ <b>Artop based</b></li> <li>○ <b>In-house</b></li> </ul> </li> </ul>	<p>IDEs used when developing may either be based on Eclipse, Eclipse and Artop or a specific one created by an AUTOSAR tool vendor. Artop implements common base functionality for tools to design and configure an AUTOSAR system (see Artop section). Artop is a layer placed on top of Eclipse and thanks to this the developer will benefit from the widely extensible plug-in system in Eclipse to get for example version control. Whether or not an in-house tool suite only focuses on what is needed for AUTOSAR and not offer the extra features made available by Eclipse makes this point an important one when comparing tool suites.</p>
<ul style="list-style-type: none"> <li>• <b>Miscellaneous</b> <ul style="list-style-type: none"> <li>○ <b>Version control</b></li> <li>○ <b>Special features</b></li> <li>○ <b>Third party integration</b></li> <li>○ <b>Safety standards</b></li> <li>○ <b>Validation</b></li> </ul> </li> </ul>	<p>Finally there are a couple of properties that do not fit into any special category but are still worth including. Version control, an aspect of software configuration management, is important when handling files that potentially could be changed at the same time by different people of a development team. Any special features making a specific vendor stand out can also play a part when making decisions on which suite to use. Often there are great tools available from third party vendors that are more commonly used in other tools or stand-alone but would be quite handy even for AUTOSAR. If that exists and the ability to integrate them into an AUTOSAR tool suite should be investigated and taken into consideration. When developing products for the field AUTOSAR is meant for safety is more often than not very important. One important property is what (if any) safety standard a tool suite uses in order to guarantee safety on the target application. It is also important to be aware of how the different AUTOSAR layers are validated by the development tools, if it is done automatically or if this is something the user has to do manually.</p>

**Table 8: Initial list of important properties for the comparison.**

	Elektrobit (EB)	Mecel	Mentor Graphics	Bosch India	Continental
Country	Finland	Sweden	USA	India	Germany
Supported AUTOSAR release(s)	3.0	N/A	N/A	3.0	N/A
License model	N/A	N/A	N/A	N/A	N/A
Price	N/A	N/A	N/A	N/A	N/A
Development (SWC, RTE, BSW)	- X X	X X X	X X X	X X X	- X X
Simulation (SWC, RTE, BSW)	- - -	X - -	X - -	- - -	- - -
Simulation platform (PC, Linux)	- -	X -	X -	- -	- -
Eclipse based IDE	X (Tresos studio)	X	X	X	X
Artop based IDE	-	-	-	-	X
In-house developed IDE	X (other)	-	-	-	-
Safety standards	N/A	N/A	N/A	N/A	N/A
Validation (SWC, RTE, BSW)	- - -	- X X	X - -	- - -	- - -

Table 9: Tool vendor comparison

	dSpace	Vector	Geensoft	ETAS	KPIT Cummins
Country	Germany	Germany	France	Germany	India
Supported AUTOSAR release(s)	2.1, 3.0, 3.1	2.1, 3.0, 3.2	N/A	2.x, 3.x	2.0, 2.1
License model	N/A	N/A	N/A	N/A	N/A
Price	N/A	N/A	N/A	N/A	N/A
Development (SWC, RTE, BSW)	X X -	X X X	X X -	X X -	- X X
Simulation (SWC, RTE, BSW)	X - -	X - -	X - -	X - -	- - -
Simulation platform (PC, Linux)	X -	X -	X -	X -	- -
Eclipse based IDE	-	-	X	-	-
Artop based IDE	-	-	X	-	-
In-house developed IDE	X	X	-	X	X
Safety standards	IEC <sup>1</sup> , ISO <sup>2</sup>	ISO <sup>2</sup>	N/A	IEC <sup>1</sup> , ISO <sup>2</sup>	N/A
Validation (SWC, RTE, BSW)	- - -	X X X	X X X	- X -	- X X

Table 10: Tool vendor comparison continued.

	The MathWorks	Arc Core	Open Synergy	pulse-AR	Infineon
Country	USA	Sweden	Germany	France	Germany
Supported AUTOSAR release(s)	N/A	N/A	3.1	4.0	N/A
License model	N/A	GPL	N/A	N/A	N/A
Price	N/A	N/A	N/A	N/A	N/A
Development (SWC, RTE, BSW)	X - -	X X X	X X X	X - -	- - X
Simulation (SWC, RTE, BSW)	X - -	- - -	X - -	- - -	- - -
Simulation platform (PC, Linux)	X X	- -	- X	- -	- -
Eclipse based IDE	-	X	X	X	-
Artop based IDE	-	-	X	X	-
In-house developed IDE	X	-	-	-	-
Safety	IEC <sup>1</sup> , ISO <sup>2</sup>	N/A	N/A	N/A	N/A
Validation (SWC, RTE, BSW)	X - -	X X X	- - -	- - -	- - -

Table 11: Tool vendor comparison continued.

<sup>1</sup> IEC61508

<sup>2</sup> ISO26262

## 5.2 Second iteration

A total of 15 tool vendors were included in the research phase and the first iteration of our comparison. As expected no tool suite managed to provide every property considered important consequently no obvious candidates emerged. Based mostly on requests by CrossControl and what we believe would simplify development of SWCs and pre/post development related tasks the original list of important properties was narrowed down and slightly changed (no single tool suite is required to offer every property but the more the better) for the second iteration:

- Complete tool chain
- Country
- Included BSW
- Linux support
- Offline SWC simulation
- Safety standards
- Validation
- Version control

Price and license model would have been interesting too but every company has to be contacted to receive this information since it was not revealed on their websites.

### 5.2.1 dSpace + EB

dSpace and EB are cooperating to be able to provide a complete tool chain [55]. We consider this a very important feature since having to deal with only one (in this case two) vendor is a lot easier than having to deal with one vendor for each step in the tool chain. As they are officially cooperating we believe they are well aware of the other party's tools as well as their own thus potentially having to deal with two vendors instead of one is not considered being a disadvantage. The dSpace part of the tool chain provides good simulation possibilities offline on a PC which makes it easier to test a system before deployment. Thanks to EB's big collection of BSW the whole dSpace + EB tool suite feels complete and a candidate worth looking further into.

### 5.2.2 Mecel

Mecel provides all the necessary tools, including BSW, in order to develop an AUTOSAR based product. If we need support it doesn't matter where in the tool chain we have a problem, one and the same company will be able to help us. Mecel is a Swedish company which means there is no risk of language misunderstandings. CrossControl has previously been in contact with Mecel regarding a different matter so they are well aware of each other and this could be an advantage for CrossControl in the future if they decide to use AUTOSAR in any of their upcoming products. Mecel's tools are based on Eclipse hence any useful plug-in typically found in Eclipse, version control to mention one, can be used during development of an AUTOSAR solution. Like most other tool suites offline PC-based simulation of SWCs is possible.

### 5.2.3 ETAS

ETAS only offer tools to create SWCs and generate RTEs. In order to get a complete tool chain including BSW modules and configuration possibilities a second vendor has to be contacted. As far as we know ETAS does not collaborate with any specific BSW provider and this can be both a disadvantage and an advantage. The downside is that if there are problems using tools unfamiliar

with each other it might take longer for the involved companies' support teams to figure out a solution. On the other hand the good thing is that you are able to choose among a handful of BSW provides and hopefully find one that suites your needs perfectly. Since there are a handful of other more promising tool suites available this one would have been discarded because of the already mentioned reasons but what makes up for the potential issues is their compliance to safety standards; the safety standards in question are ISO 26262 and IEC 61508 [56]. This is something believed to be of importance for CrossControl considering their field of interest. All SWCs can be simulated on a PC to ensure proper behavior before they are downloaded onto the intended target ECU(s).

#### **5.2.4 Arc Core**

Arc Core is a company offering a complete tool chain called Arctic Core as well as BSW; so if CrossControl in the future decides to use AUTOSAR in some of their products Arc Core's tools have the means to help out in all areas of the development. Arctic Core is based on Eclipse like many other tool suites thus version control is available if desired. Another advantage is good validation possibilities throughout the tool chain with the purpose of ensuring everything is done properly. It is a Swedish company therefore communicating with them is less likely to be a problem. Arc Core is the only tool vendor we discovered that explicitly states the license model on the website; all the others have to be contacted. Arctic Core is released under the GPL which means it is open-source; it is however possible to obtain a commercial license which removes the obligation to release your developed products under the same license. Also important to know is the fact that the commercial license includes Technical support from Arc Core's dedicated support team.

#### **5.2.5 Vector**

Vector has an in-house developed tool suite covering all the steps in a typical AUTOSAR tool chain. Not only this, a very comprehensive collection of BSW modules is also available. This completeness and their well-established status in the industry make Vector a very interesting candidate worth looking further into. Developed SWCs can easily be tested on a PC-based environment before being set up on the intended hardware.

#### **5.2.6 Open Synergy**

Open Synergy has the only Linux-based solution of the vendors we have included in our comparison; in fact the only Linux solution we found while doing research. The tools allow you to implement, run and test SWCs directly on top of a Linux operating system as well as generating an RTE interfacing the SWCs and the BSW. Since CrossControl are developing Linux-based x86 display computers Open Synergy's solution is of great interest as one of its typical areas of use resembles the display computer, namely an x86 car-PC.

### **5.3 Result from first and second iteration**

As can be seen in the previous sections the initial list of 15 vendors was narrowed down to six. All of them were sent inquires to evaluate their software and below the result is presented.

#### **5.3.1 dSpace + EB**

In order to use tools from dSpace and EB both companies needed to be contacted separately even though they will be used together as one. dSpace were interested in providing us with the possibility to test their tool for graphical development of SWCs but unfortunately EB never responded thus

dSpace was no longer an option. CrossControl did not place much importance in testing the graphical part of development although we felt it would have been interesting to have that opportunity.

### **5.3.2 Mecel**

Mecel did respond to our inquiry about evaluating their tool suite and presented two options:

- Do an evaluation and compare their tool suite with three tool suites of their competitors and provide them with the final results.
- Pay for a complete AUTOSAR evaluation kit including hardware ready to be used as it is.

Accepting the terms of the first option was no problem but they wanted to have their upcoming version evaluated and we would not be able to get it until about half way through our thesis. As they could only provide an estimated release date this option was not working. The second option was rejected due to financial reasons.

### **5.3.3 Etas**

They never responded to our inquiry.

### **5.3.4 Arc Core**

ArcCore were one of the first companies to respond and accept our inquiry. Arctic Studio was one of the tool suites chosen to be evaluated.

### **5.3.5 Vector**

Unfortunately they never responded to our inquiry; they seemed to have the most comprehensive tool suite hence it would have been very interesting to evaluate it.

### **5.3.6 Open Synergy**

Open Synergy were quick to respond and offered us an evaluation license of their tool suite. Their tools are the only ones in our comparison with Linux support and since that is of interest to CrossControl the opportunity to try it out was very important. In other words, COQOS-tP was the second tool suite chosen to be used for developing the demonstrator.

## 6 Demonstrator

To be able to answer some of our questions in section 1.2, evaluate AUTOSAR and to draw proper conclusions a demonstrator had to be created. The following sections will present the demonstrator created using Arctic Studio and COQOS-tP which are the selected tools from the previous chapter.

### 6.1 Requirements of the demonstrator

In order to be able to answer some of our questions in section 1.2, our goal has been to design and implement a system which fulfills the following requirements:

- **Realistic** – The system shall be designed as realistic as possible, preferably functionality that already can be found in an ordinary car.
- **Simple** – The functionality of each part in the system and as a whole shall be easy to understand, regardless of readers' previous AUTOSAR experience. Yet the system should be complex enough to make proper conclusions about realistically sized systems.
- **Several SWCs** – The system shall be designed to use several SWCs in order to test migration, reusability and other AUTOSAR features.
- **Several Runnables** – The system shall be designed to use multiple ECUs and execute more than one runnable/ECU in order to test scheduling and OS services.
- **Communication** – The system shall be design to use inter-communication over a bus, e.g. CAN and intra-communication between SWCs.
- **Input/Output** – The system shall be designed to receive external signals (buttons) and send signals to external components (LEDs) in order to test I/O functionality.

### 6.2 What we have done

The best way to actually test how the AUTOSAR standard is applied in practice is to model a life-like system and create it with available tools. Since the standard is an initiative from the automotive industry we decided to extend that initiative and create a system that could be found in an ordinary car, to be more precise, a seat heating system.

We have captured what we consider to be necessary functionality of such system:

- The ability to decide how warm the seat should be – Input.
- Discovery if a person is sitting on the seat – Input.
- Control the heating element – Output.

These three parts should then be controlled by a supervising SWC, independent of the actual location of the other components which handles the above list. This system is what we consider simple in the manner of functionality, yet complex enough to fulfill our system requirements. In Table 12 a summary of the requirements of the demonstrator along with some of made system's properties is available.



	Seat Heating system	Five SWCs used	Reused SWCs	External input	Output to Seat Heater	Transferable SWCs	C-S and S-R Interfaces	Intra/Inter com
Realistic	X		X	X	X	X	X	X
Simple	X	X						
Several SWCs		X				X		
Several Runnables		X				X		
Communication						X	X	X
Input/Output				X	X			

Table 12: Requirements and the actual system

How the SWCs can and should be connected can be seen in Figure 14 which is the VFB view over the system. Some of the properties found in Table 12 can be seen in this figure.

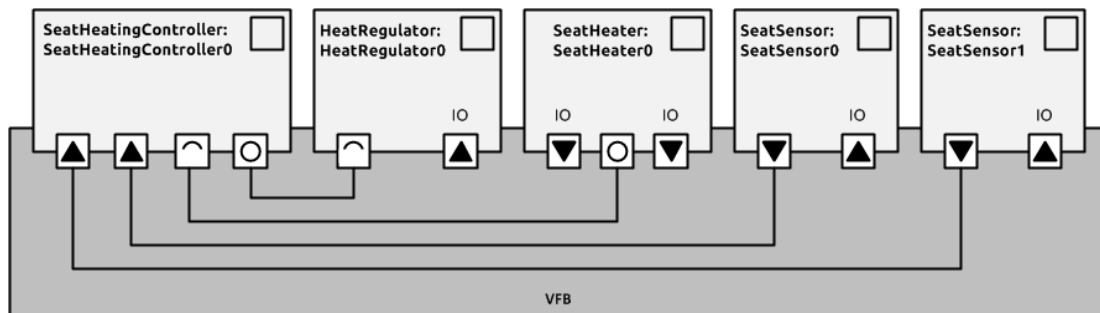


Figure 14: VFB view of the system

## 6.3 Help from suppliers

During our thesis we have had continues contact with three different companies during different stages of the development. In this section, each company will be briefly presented and how we received help from them.

### 6.3.1 Workshop at ArcCore

We have used the Artic Studio IDE from ArcCore during our thesis project, to create AUTOSAR solutions for our development board. They have been very friendly and given us much help, not just about the IDE but also AUTOSAR in general. Most of our problems and questions have been solved via email, but this is not the ideal communication medium, so in order to help us even better, they could offer us a half-day workshop at their headquarters in Göteborg, a chance we gladly accepted.

During one of our telephone meetings we discussed why other companies struggle with the implementation of AUTOSAR solutions; component based development has been around for quite a while. According to our contact person at ArcCore, the problem is quite common. A company read about the standard and understands it, just like us before the workshop. Then when it is time to start creating solutions, the struggle begins. Once you have passed the toughest part, it is quite easy to implement AUTOSAR solutions, according to ArcCore.

The first meeting we had with ArcCore, we both traveled to Göteborg to discuss AUTOSAR in general and also have a workshop. We worked with one of their employees for one afternoon. Our focus was to know more about how their tool was intended to be used. What files you as a user are suppose to create by your own etc. It was very giving and interesting on several levels. Just too actually meet the people you previously only have had email contact with. We also got in touch with some other employees, which we have received help from after the workshop. Depending on the topic or problem, we knew who to ask what.

We also had a second meeting, but that time only Jesper visited them since he was in Göteborg over a weekend. That meeting was also good, but their guy that is working with the STM32 was away that day, so we could not ask him questions directly. Yet, Jesper had the opportunity to discuss other matters with them and even report an error found in the tool.

### **6.3.2 Open Synergy**

Open Synergy was contacted by e-mail plenty of times in order to ask for advice regarding user interaction with the demonstrator and solutions to problems such as generator error outputs. For more complex issues such as the communication stack our contact person at Open Synergy called us up in order to more easily understand our situation and provide a possible solution. We only have good experiences from contacting Open Synergy.

### **6.3.3 IXXAT**

IXXAT support was very helpful when problems arose when using CAN together with COQOS-TP. After several weeks of e-mail conversations our contact person managed to solve our problems. As with Open Synergy, based on our experiences only good things can be said about IXXAT's support staff.

## **6.4 Design decisions**

Once the requirements were in place it was time to start thinking about how to come up with a design that was able to fulfill these requirements. This section describes important design decisions that had to be made early in the designing phase.

### **6.4.1 Basic software modules used**

Currently there are about 60 modules [57] with defined interfaces and behavior. All these modules are not available in the tools we have used, the tool vendors cannot keep up with the pace when new modules are defined. It would have been very complex and time consuming if we would have designed a system that utilizes all available modules. It is our belief that the examination of fewer modules more thorough outweighs a less truthful examination of more modules. Therefore we made a decision to only use the modules needed to fulfill our system requirements, mainly I/O and COM.

### **6.4.2 Only Sender-received interfaces between nodes**

One cornerstone in the standard is the ability to place your SWCs on different ECUs without worrying about the actual network topology. There should be no restrictions when deciding if the SWCs should use a sender-receiver or client-server interface. Depending on the situation, one or the other might be a better choice. During our first meeting with ArcCore we discussed this and we were told that you are actually restricted when it comes to choose a proper interface for your SWCs.

According ArcCore, no tool vendors have successfully created a tool that can generate a system that handles Client-Server interface between two ECUs separated by a bus. We have not been able to

confirm this issue with other tool vendors, but we can see the cause of the problem. A Client-Server interface is realized as function call from one SWC to another SWC with the implementation of the function. If these components are located on the same ECU using the same memory, it is okay. However, if they are separated by the bus it is complicated to handle it. It would be necessary to set up some kind of channel between the two ECUs and that is not covered by the current standard.

It is unclear if COQOS-tP has support for this or not since their user manual is ambiguous on this matter [58].

We should be able to design the system regardless of the system topology, but we are in practice restricted to only use sender-receiver interfaces over buses.

***This answers Q1.13 “Does the standard propose something which in practice is limited?”***

The standard states that you are free to map the SWCs to any node without taking interfaces into account. However, when this is realized limitations are present.

## 6.5 Communication

Our motivation was to create a demonstrator system that could help us answer some of our questions.

- How easy or hard is it to implement and configure a working system with our tools?
- Which parts of the created AUTOSAR configure files can be shared between the tools, is it as easy as the standard wants it to be?

The simplest type of system is to run a single SWC on one ECU with no external communication or stimuli, but that is not how the AUTOSAR standard is intended to be used. The standard should be applied to a more complex system that involves both intra-, inter-communication and external signals.

The BSW layer is as earlier mentioned divided into several categorized modules, and depending how you watch them; they could be seen as into horizontal (layer) and vertical categories (stack). There are especially three stacks in the BSW that we are interested in:

- **System stack** – Includes System Services.
- **Communication stack** – Includes COM services, COM Hardware Abstraction and COM Drivers.
- **I/O stack** – Includes I/O Hardware Abstraction and I/O Drivers.

### 6.5.1 System stack

AUTOSAR offers a bunch of System Services that other modules can use and an entire AUTOSAR OS. The OS is needed to schedule runnables so we are not only motivated to use and test it; it is a requirement to run AUTOSAR applications on an ECU. We wish to learn and know more about the configuration possibilities of the AUTOSAR System stack, especially the OS. As stated earlier in the design decisions, we are not going to use all system service modules.

### 6.5.2 Communication stack

One of our requirements of the system was to use inter-communication between two or more ECUs. We must understand the dataflow and how to configure the system to use a bus, in our case a CAN-bus, to be able to communicate between ECUs. We are motivated to learn more how to configure and use communication since it is widely used in modern cars.

### 6.5.3 I/O stack

A big motivation for us was to create a life-like system using an evaluations board with bus connectors, buttons and LEDs. AUTOSAR is meant to be used in automotive environment which usually always involves sensors and actuators, and we wanted to mimic that. One possibility was to create applications with simulated external events (switches, buttons etc) since the applications are decoupled from the hardware and doesn't know where the data is coming from. However, that would violate our own criteria for the system and would not utilize the standard in a good way. It would especially be interesting to see how much easier or harder it would be to configure the peripheral-access rather than implement the functionality.

## 6.6 Workflow

A workflow was setup early in the development phase of the demonstrator; it can be seen as a number of steps identifying tasks that need to be carried out in a particular order with the aim of producing a successful outcome (see Figure 15).

The first step involves identifying the problem for the demonstrator to solve. Once the problem is known the process of designing a system based on this information can commence. Relationships between SWCs in the system design has to be determined due to the limitation introduced when communication between nodes (see Only Sender-received interfaces between nodes, section 6.4.2). When it has been established how SWCs communicate with each other this information can be used to group SWCs requiring to be placed on the same node. The next two steps are about deciding what nodes to use and how the groups are distributed among them. Once it is known on what node each group is placed it is possible to determine how many signals are needed in order to communicate between nodes. Before the last step of verifying that the demonstrator has solved the problem that was identified in the first step the hardware has to be configured and prepared for the software.

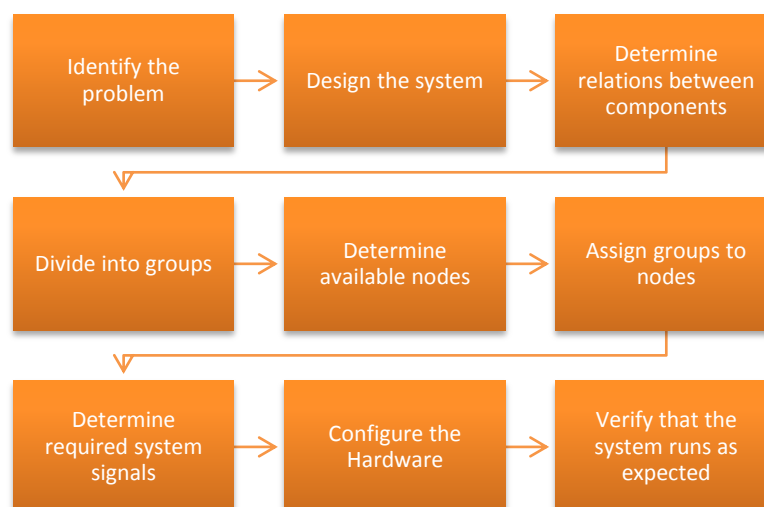


Figure 15: Workflow during the realization of the demonstrator.

## 6.7 Equipment

In order to realize the demonstrator a set of software was needed for different areas of the development. In this section a description of each tool will be presented (see Table 13). The demonstrator implementation created by each tool suite was run on different hardware depending on the requirements; the chosen hardware is presented in more detail below as well (see Table 14).

### 6.7.1 Used Software and Hardware

Software	Version	Note
<b>IAR Embedded Workbench for ARM 6.10 Kickstart</b>	6.10	Used to relearn about STM32
<b>STMicroelectronics STM32 ST-Link Utility</b>	1.2	Loader for .bin files
<b>Atollic TrueSTUDIO Lite</b>	1.4.0	Used for debugging
<b>Keil uVision 4</b>	4.13a	Used to learn about STM32
<b>ArcCore Arctic Studio</b>	1.3	AUTOSAR IDE for STM32
<b>ArcCore RTE Builder</b>	1.2.1	
<b>ArcCore ECU Extract Builder</b>	1.1.0	
<b>ArcCore BSW Builder</b>	1.2.1	
<b>ArcCore SWC Builder</b>	1.2.1	
<b>IXXAT Minimom</b>	2.18 & 3.3	CAN-Bus activity
<b>WinMerge</b>	2.12.4.0	File comparison
<b>Nokia QT Creator</b>	2.0.1	Used to create I/O Sim.
<b>Artop</b>	2.1	
<b>OpenSynergy COQOS RTE Generator</b>	2.3.0	Generator for Linux
<b>OpenSynergy COQOS BSW Generator</b>	2.3.1	Generator for Linux
<b>Gedit</b>	2.30.3	

Table 13: Software used when developing the demonstrator

Hardware	Note
<b>STMicroelectronics STM3210E-Eval</b>	Evaluation board
<b>STMicroelectronics ST-Link</b>	Debugger
<b>IXXAT USB-to-CAN II</b>	CAN-Adapter
<b>IXXAT USB-to-CAN Compact</b>	CAN-Adapter
<b>Compaq Evo N600c</b>	Laptop

Table 14: Hardware used when developing and running the demonstrator

## 6.8 Nodes

The two types of hardware used by the nodes in the system are described in this section.

### 6.8.1 STM3210E-Eval

ArcCore recommended us to not use boards with too little RAM and preferably a microcontroller they had support for in their tools and the result was the STM3210E-Eval board from STMicroelectronics[59].



Figure 16: STM3210E-EVAL board. Courtesy of STMicroelectronics

The STM3210E-Eval board was selected to be used during the thesis based on these properties:

- **Enough RAM and Flash** – With a total amount of 64 KB RAM and 512 KB Flash, the board passed the recommendations by ArcCore (64 KB RAM and 128 KB Flash).
- **I/O and Peripherals** – The board have 4 LEDs, a four-direction joystick, buttons and one CAN connection. The LEDs are possible to use for output, the joystick and buttons as input and CAN for network communication.
- **Based on ARM Cortex-M3** –. The processor is interesting for CrossControl and is one of the main reasons why this board was selected.
- **Tool support by ArcCore** - At the start of the thesis, ArcCore did not have support for this processor in their stable versions. They were going to add support in January 2011, which they also did.

### 6.8.2 Laptop with Ubuntu 10.04

At the time of the thesis neither of CrossControl's display computers was available to be used so instead a decision was made to set up a Linux environment on regular computer instead, in this case a laptop. COQOS-tP had the following requirements:

- Linux Ubuntu x86 9.10 or newer.
- GCC 4.x
- JDK (Artop IDE)

To allow input and display output on this node a graphical application was developed to facilitate similar functionality as the physical buttons and LEDs etc. available on the STM3210E-Eval board (see section 6.11).

## 6.9 System

In order to test the chosen tool suites an AUTOSAR system was modeled. When creating the model, a real world scenario was kept in mind as it felt more relevant to do something you would typically use AUTOSAR for, instead of making something up. As a result we ended up with a model of a seat heating system (see Figure 17: Seat heating system) which is relevant for both the automotive industry (AUTOSAR's main target) as well as CrossControl's target industry. The model consists of four SWCs (one is instantiated two times) and two port interface types (Client-Server and Sender-

Receiver). Each SWC has a specific purpose and with the intention of testing the relocatability feature of AUTOSAR we will be placing them on different ECUs and in a number of ways when it is time to realize the system.

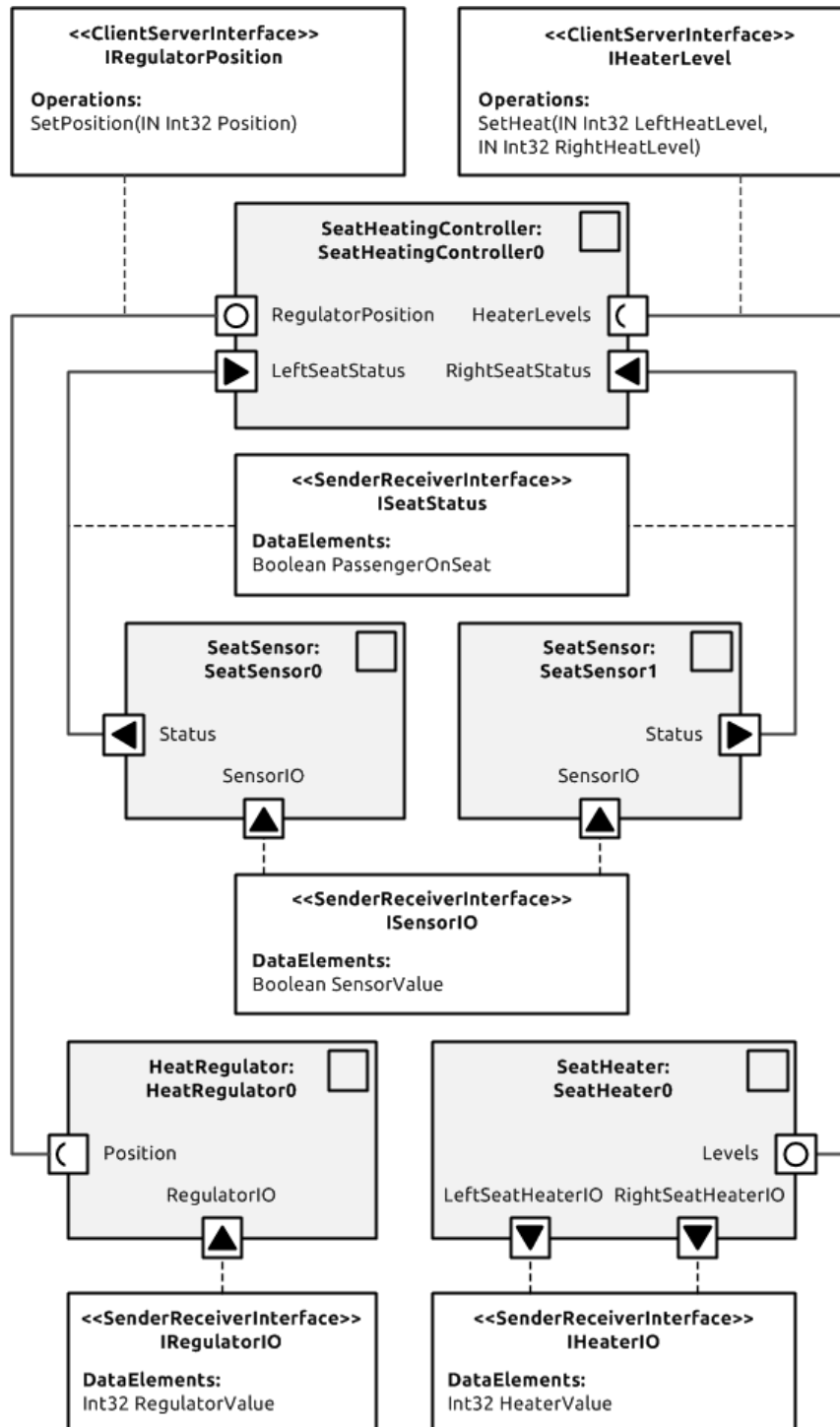


Figure 17: Seat heating system

### 6.9.1 Components

In this section, the used SWCs will be presented briefly.

- **SeatHeatingController** - This component is an application SWC and the main component in the model. All decision-making is performed by this component based on input from the SeatSensor0, SeatSensor1 and HeatRegulator0. Output is sent to the SeatHeater0 which changes the heating element accordingly.
- **SeatHeater** - The SeatHeater SWC is controlling the seat heating element. It is controlled by the SeatHeatingController if the heating of the seat shall be changed (Off, Low or High).
- **SeatSensor** - This component has the purpose of detecting whether or not someone is sitting on the seat. The status of the seat is sent to the SeatHeatingController.
- **HeatRegulator** - The HeatRegulator component is keeping track of the position of the physical seat heating regulator dial. This position is used by the SeatHeatingController in order to know if the heat should be decreased, increased or remain the same..

### 6.9.2 Runnables

Each SWC contains one or several runnables which defines the behavior of the component. These can be triggered either by a Timing Event or an Operation invoked event. When a runnable is triggered by a Timing Event, it is scheduled with a specific period time and tasks must be created which the runnable can be mapped to. With Operation Invoked Event can be compared with function calls. The runnable is triggered when another SWC calls the runnable through a Client-Server interface.

So called *Inter Runnable Variables* can be used to communication between Runnables within a SWC and can be compared with global variables In this system, only one of the SWCs use Inter Runnable Variables (SeatHeatingController).

All the Runnables contained by the SWCs used in the system, can be found in Table 15.

Component	Runnable	Event type	Inter Runnable Variable
SeatHeatingController	SeatHeatingControllerMain	Timing	InterRunPositionVar (Read)
	SeatHeatingControllerSetPosition	Operation invoked	InterRunPositionVar (Write)
SeatHeater	SeatHeaterMain	Operation invoked	
HeatRegulator	HeatRegulatorMain	Timing	
SeatSensor	SeatSensorMain	Timing	

Table 15: Runnables for each SWC

### 6.9.3 Port interfaces

In this section, each Interface used by the system will be briefly explained. See section 2.5.2 for an introduction of Interfaces.

#### 6.9.3.1 IHeaterLevel

This is the client-server port interface between the *SeatHeatingController* component and the *SeatHeater* component. The PPort Levels of the *SeatHeater* component provides *SeatHeatingController* with a method called *SetHeat* which is used to set the heating element for the left and right seat.



### 6.9.3.2 IHeaterIO

*SeatHeater* has two PPorts which share the same Sender-Receiver Interface, namely *IHeaterIO*. This interface defines a 32 bit integer value called *HeaterValue* which is supposed to be used by an actuator to set the heat level on a heat element. The actuator is simulated on the Laptop and on the STM32 board; LEDs are used to indicate the used heat level.

### 6.9.3.3 ISeatStatus

The *SeatController* is a sender-receiver port interface used by the *SeatSensor* component in order to provide the *SeatHeatingController* component with a boolean data element indicating the seat status.

### 6.9.3.4 ISensorIO

This Sender-Receiver Interface defines a Boolean value called *SensorValue*. This value should be written by a sensor indicating if a person is sitting on the seat or not. The sensor is simulated on the Laptop node and on the STM32 board node, buttons are used as sensors.

### 6.9.3.5 IRegulatorPosition

This client-server port interface provides the *SeatHeatingController* with the ability to retrieve the physical position of the regulator dial from the *HeatRegulator* component which will be used to set the heating element accordingly.

### 6.9.3.6 IRegulatorIO

The SWC *HeatRegulator* has one RPort with a Sender-Receiver Interface called *IRegulatorIO* which contains a 32-bit integer value called *RegulatorValue*. This data element should be written by a sensor which reads the desirable heat level. This sensor is also simulated on the Laptop node and buttons are used on the STM32 board node.

## 6.10 Topology cases

For the communication between our five SWCs, we have four different connections. These connections are summarized in Table 16 and can also be seen in Figure 14. At this stage we are interested to know how the SWCs should be connected and with what type of interface. Depending on the type of communication, inter or intra, system signals must be created in later steps to support the topology.

From	Port	Interface	Port	To
SeatSensorRight	Status	S-R <sup>3</sup>	RightSeatStatus	SeatHeatingController
SeatSensorLeft	Status	S-R3	LeftSeatStatus	SeatHeatingController
HeatRegulator	Position	C-S <sup>4</sup>	RegulatorPosition	SeatHeatingController
SeatHeatingController	HeaterLevel	C-S4	Levels	SeatHeater

Table 16: Available Connections

We have then divided our five SWCs into three different groups, see Table 17. As we have discussed earlier, only sender receiver interfaces are possible to use for inter communication between nodes. These groups are color coded to make it easier for the reader to see which components must be placed together on the same ECU.

<sup>3</sup> Sender-Receiver

<sup>4</sup> Client-Server

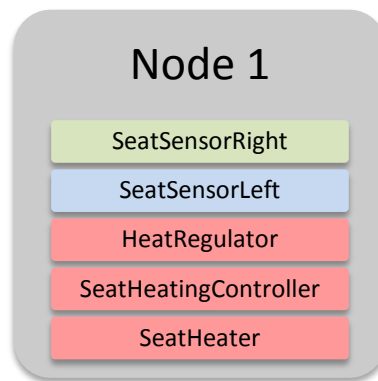
SWC	Group
SeatSensorRight	Group A
SeatSensorLeft	Group B
HeatRegulator	Group C
SeatHeatingController	
SeatHeater	

**Table 17: Groups of SWCs**

Both SeatSensorRight and SeatSensorLeft have interfaces of type Sender-Receiver and are therefore divided into two different groups (A and B); they are possible to place on different nodes in the system. The remaining SWCs: HeatRegulator, SeatHeatingController and SeatHeater must be placed on the same node since they share information through Client-Server interfaces. They are therefore gathered together in group C.

Now when the SWCs are divided into different groups and we know how they are connected together, it is possible to determine the necessary system signals for each different case.

#### 6.10.1 Case 1 – All SWCs on the same node

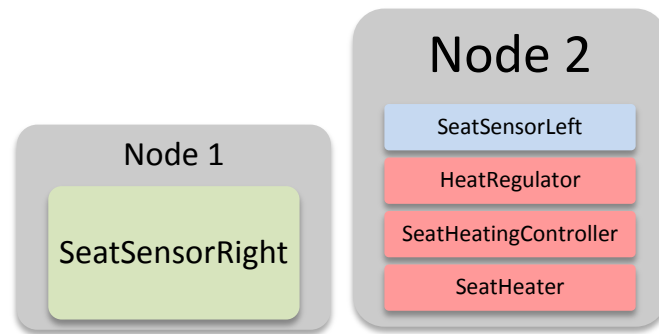


In this case, all the available SWCs are mapped to the same node. No communication with other nodes means that only intra communication will be present, see Table 18. This case could be considered as the simplest one since no configuration of the communication modules must be done with the tools.

SoftwareComponentName.Port	Communication
SeatSensorRight.Status	Intra
SeatSensorLeft.Status	Intra
HeatRegulator.Position	Intra
SeatHeatingController.HeaterLevel	Intra
SeatHeatingController.RightSeatStatus	Intra
SeatHeatingController.LeftSeatStatus	Intra
SeatHeatingController.RegulatorPosition	Intra
SeatHeater.Levels	Intra

**Table 18: Communication between SWCs on one node**

### 6.10.2 Case 2 – One SeatSensor SWC mapped to an additional node



In this case, group A is extracted from the node where group B and C reside and placed into a separate node. This is possible since communication from group A to group C is based on Sender-Receiver interfaces. Now there is not only intra but also inter communication and system signals must therefore be introduced. As we can see in Table 19, two system signals must be created. One for Node 1 with the direction set to send and one for Node 2 with the direction set to receive. For this type of setup, two nodes must be configured. Our motivation for extracting group A is the following:

- Extracting group A or B would be the same since both groups contain the same SWC, just different instantiations of it. The functionality of these two groups is therefore the same.
- Extracting group C from group A and B would be possible, but Case 2 and Case 3 would almost be identical. The reason for this is that the same amount of system signals had to be created. The only difference between Case 2 and Case 3 would be that two nodes are used instead of three.

SoftwareComponentName.Port	Communication
SeatSensorRight.Status	Inter
SeatSensorLeft.Status	Intra
HeatRegulator.Position	Intra
SeatHeatingController.HeaterLevel	Intra
SeatHeatingController.RightSeatStatus	Inter
SeatHeatingController.LeftSeatStatus	Intra
SeatHeatingController.RegulatorPosition	Intra
SeatHeater.Levels	Intra

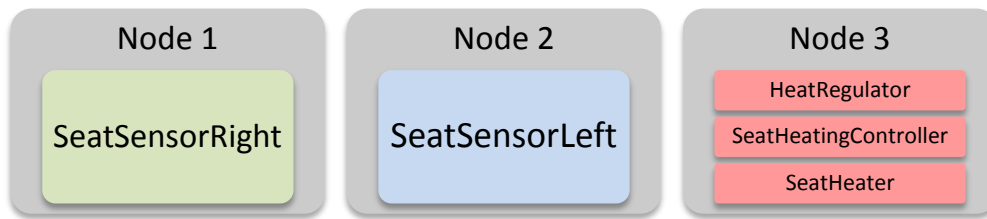
Table 19: Communication between two nodes

Since we at this stage of the workflow know that we must create two system signals, it is wise to decide some of the common parameters for the configuration of system signals, such as ID, Size etc. See Table 20.

Direction	Signal	PDU	PDU Size	CAN ID	CAN Type	Baudrate
Transmit	SeatSensorRightTx	SeatSensor	16	1	Extended	125
Receive	SeatSensorRightRx	SeatSensor	16	1	Extended	125

Table 20: Common parameters for the system signals with two nodes

### 6.10.3 Case 3 – All groups separated



The last case could be considered as the most complex since it involves most nodes and most system signals of all the three cases. Most configurations must be done in this case and the topology also affects how the BSW modules on each node must be configured. In general it means that for each node you add to the system, additional and different configuration must be made in the modules. This is not that surprising, but still, it has to be done and could increase the chances of miss configuration of the different nodes. In Table 21 we can see which SWCs that will use inter communication.

SoftwareComponentName.Port	Communication
SeatSensorRight.Status	Inter
SeatSensorLeft.Status	Inter
HeatRegulator.Position	Intra
SeatHeatingController.HeaterLevel	Intra
SeatHeatingController.RightSeatStatus	Inter
SeatHeatingController.LeftSeatStatus	Inter
SeatHeatingController.RegulatorPosition	Intra
SeatHeater.Levels	Intra

Table 21: Communications between three nodes

Just like Case 2, we know at this stage which system signals we must create in order to use the selected topology. This is very helpful and should be used to make the configuration process smoother. Like the previous case we have decided some of the common properties for these signals, see Table 22 for a complete overview of the signals.

Direction	Signal	PDU	PDU Size	CAN ID	CAN Type	Baudrate
Transmit	SeatSensorRightTx	SeatSensor	16	1	Extended	125
Transmit	SeatSensorLeftTx	SeatSensor	16	2	Extended	125
Receive	SeatSensorRightRx	SeatSensor	16	1	Extended	125
Receive	SeatSensorLeftRx	SeatSensor	16	2	Extended	125

Table 22: Common parameters for the system signals with three nodes

### 6.10.4 Summary of cases

	Case 1	Case 2	Case 3
Nr of Nodes:	1	2	3
Nr of Signals:	0	2	4

Table 23: Summary of nodes and signals

Depending on how many nodes and signals that are involved in the complete system, the configuration of each node will differ and these variations must be taken into account. More tasks

must be created if more runnables are supposed to reside on one node (depending on the requirements and timing properties).

## **6.11 Mapping of groups to nodes**

Now that the SWCs have been grouped together they can now be mapped to nodes based on the cases defined in the previous section.

### **6.11.1 Case 1**

In case 1 where all groups are placed on the same node two scenarios existed. In the first one all nodes were placed on the STM32 board and in the second one the same placement was done on the Linux laptop.

This case allowed parallel execution of two complete systems and it was easy to verify that the same input on both systems yielded the same output.

### **6.11.2 Case 2**

When case 1 was reconfigured to case 2 group A was lifted out onto its own node; in case 3 the same action was also performed for group B thus case 2 and 3 are very similar. This led to the conclusion that case 2 does not provide enough useful data compared to if only case 1 and 3 were realized in order to justify it; hence case 2 was considered redundant and was eventually discarded.

### **6.11.3 Case 3**

Case 3 is basically the opposite of case 1; instead of placing all groups on the same node each one is placed on a separate node. There were multiple combinations available in regard to which hardware to put which group on but the following one was chosen:

- Group A → STM32 board
- Group B → STM32 board
- Group C → Linux Laptop

Either combination would have worked equally well but since both group A and B consist of a seat sensor it felt natural to put them on the same type of hardware; namely the two STM32 boards. Also it was preferred to display the output graphically on the Linux laptop thus this placement seemed like the perfect fit especially as the intention never was to try all possible combinations.

Except for verifying the expected behavior by looking at the input dependent output of the system IXXAT's Minimon application was used to monitor the traffic between the nodes.

## **6.12 I/O Simulator**

The demonstrator uses input in order to calculate output values. When it runs on the STM32 board it can make use of the physical buttons and LEDs located on the circuit board to get input and provide output. However if the demonstrator is instead placed on a Linux computer such I/O possibilities are no long available thus a different way of interacting with the system has to be developed.

### **6.12.1 Functional requirements**

This I/O simulator is rather simple and only includes the necessary features needed to provide the input and display the output from the demonstrator. A list of functional requirements was created in order to assert that no necessary functionality was forgotten (see Table 24).

Id	Requirement	Description
<b>IOS-UI1</b>	Adjustable heat level.	It shall be possible to adjust the heat provided to the seats.
<b>IOS-UI2</b>	Independent left/right seat toggling of passenger detection.	It shall be possible to individually toggle whether or not the left and right seat is currently occupied.
<b>IOS-UI3</b>	Display individual heat of the left and right seat.	It shall be possible to see the current heat of both seats independently of one another.
<b>IOS-COM1</b>	Ability to transmit inputted data to the demonstrator.	It shall be possible to send heat level and seat status input to the demonstrator.
<b>IOS-COM2</b>	Ability to receive outputted data from the demonstrator.	It shall be possible to receive the calculated heat of each seat from the demonstrator.

**Table 24: Collection of functional requirements, UI and communication related.**

### 6.12.2 Realization

Fulfilling the requirements defined for the I/O simulator on Linux involves two steps: creating the graphical user interface (GUI) and the communication. Using the STM32 board an additional SWC (SeatHeaterIOBridge) along with some extra implementation was needed instead.

The behavior of the I/O simulator is determined completely by its heat dial input and output from the demonstrator. Heat is provided in the form of integer values in the interval of zero to two; zero means no heat is set and two is the warmest setting the demonstrator uses. A seat sensor will detect whether or not the seat is vacant, if it is, zero is set otherwise one. Output from the demonstrator will be within the same interval as the heating dial and interpreted as can be seen in Table 25. A hollow circle means that the LED is lit while a filled circles means it is not.

Heat dial	Seat sensor	LCD output (Linux)	LEDs (STM32)
<b>0</b>	0	OFF	●●
	1		
<b>1</b>	0	OFF	●●
	1	LO	○●
<b>2</b>	0	OFF	●●
	1	HI	○○

**Table 25: Behavior of the output depending on the input.**

#### 6.12.2.1 Linux

##### 6.12.2.1.1 Graphical user interface

When developing a GUI application there is one important question to ask oneself, is it easier to make your own customized GUI controls or try to integrate an existing GUI library? Sometimes it can be overkill to integrate a big GUI library especially if only a few controls are needed; then it could be a good idea to make your own controls for this particular situation. But since time may be a factor as well it could end up taking longer to develop a few controls than to integrate one of the libraries on the market; especially if the developer is not used to this type of programming.

For the I/O simulator a decision was made to use one of the existing libraries on the market. The reasons for this decision are we don't have a complex application we need to integrate the library

into which may possibly have caused problems and delays. In addition, using libraries that have been used by a lot of people for a long time are less likely to suffer from immature symptoms than a customized implementation just for this application.

Once a decision has been made to use an existing library one question remains, which of the available libraries to use?

#### 6.12.2.1.1.1 Technology

Two libraries were considered for the GUI, Qt [60] and wxWidgets [61]. Not much effort was put into the selection of which one to use since they provide very similar functionality but eventually Qt was selected based on previous experience and the fact that it comes with a specialized IDE.

#### 6.12.2.1.1.2 Implementation

In order to fulfill requirement IOS-UI1 and IOS-UI2 it was necessary to find controls that would allow input expected by the behavior description and if possible at the same time limit or exclude unwanted input completely. Text boxes are for example prone to give unwanted input as the user is only limited to the characters of the keyboard unless a complex algorithm is created to filter out unwanted key presses. For this reason it would be a good idea to use for example a slider control; here you can specify an acceptable interval and only to use integer values. An even better alternative would be to use a dial/knob control, it is basically the same as a slider control but a knob looks more appropriate for a heat adjuster. In the case of inputting a true or false status, which is what a seat sensor essentially does, a non-tristate check box is the perfect fit. The user can change the state of the check box depending on whether the seat is occupied or not.

For requirement IOS-UI3, displaying the received seat heat calculated by the demonstrator, some kind of display control would be an ideal fit. The simplest control for displaying text is the label but Qt provides a more interesting control called LCD number. It is a bit limiting of one's intention is to display other characters than digits but every letter that can be displayed on an seven-segment display can also be displayed by this control. As can be seen in Table 25 the I/O simulator will display the different heat levels in the form of OFF, LO and HI; this can be displayed by the LCD number control by assigning the following values: OFF, LO and H1 respectively.

With the previous argumentation in mind a knob control was chosen for the heat dial and a check box- and LCD number control for each seat in order to change seat sensor status and display the current heat. The outcome can be seen in Figure 18.

See Appendix 5: Linux I/O Simulator source code.

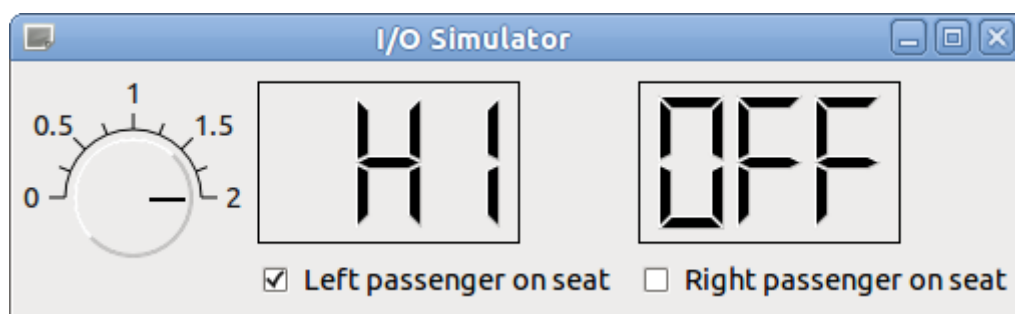


Figure 18: Example of the graphical user interface when the heat dial is set to max and a passenger is sitting on the left seat.

It was noticed that when looking for a control to handle the heat dial there were limitations of the dial control of Qt. After along time of trying without successfully enabling labeling the Qwt [62] add-on of Qt was installed. Qwt is a GUI component extension for technical applications which include among other things an implementation of a dial control called knob. Using the knob control we managed to get the desired result.

#### 6.12.2.1.2 Communication

In order to make use of the I/O simulator a way to communicate with the demonstrator had to be established. After a discussion with Open Synergy regarding the best way to handle user interaction with an AUTOSAR system he recommended the use of sockets which is what eventually was used.

##### 6.12.2.1.2.1 Implementation

The two most commonly used protocols when communicating by means of sockets are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP is a stream socket type which is connection oriented and guarantees that data sent is received on the receiving side. UDP is a simpler connectionless socket type which uses datagrams to send data between the sender and receiver. For the task at hand the latter is adequate enough especially since all the communication is done on the local machine.

Two sockets are used, one for incoming data (requirement IOS-COM2) and one for outgoing data (requirement IOS-COM1). When data is received on the incoming socket an event is triggered telling the simulator a new value is ready to be fetched. In the opposite direction it is instead the user that triggers send events by changing the heat dial or changing the state of a seat sensor for example.

##### 6.12.2.1.2.1.1 Datagram

All data sent between the simulator and the demonstrator is in the form of a structure called Datagram (see Table 26).

Data	Description
<b>Event</b>	The event (see Table 27) causing the Datagram to be sent in the first place.
<b>SeatStatus</b>	If the current event is a seat status change this field holds the new status of seat in question.
<b>HeatingDialPosition</b>	New heating dial position in case of a heating dial change event.
<b>SeatHeat</b>	Holds information about the new seat heat of a particular seat when the event field is set to a seat heat change event.

Table 26: Data fields of the Datagram structure.

Events	Description
<b>LeftSeatStatusChanged</b>	Triggered by the simulator when the status of the left seat has changed.
<b>RightSeatStatusChanged</b>	Triggered by the simulator when the status of the right seat has changed.
<b>LeftSeatHeatChanged</b>	Triggered by the demonstrator every time a new heat level has been calculated for the left seat.
<b>RightSeatHeatChanged</b>	Triggered by the demonstrator every time a new heat level has been calculated for the right seat.
<b>HeatingDialChanged</b>	Triggered by the simulator when the heat dial has changed.

Table 27: Possible events causing a Datagram to be sent.



### 6.12.2.2 STM32

Different actions had to be carried out to fulfill the same requirements when using the STM32 board. In this section the needed actions will be presented.

#### 6.12.2.2.1 Peripherals

With the STM32 board, it is possible to use physical buttons to generate input to the system and use LEDs to represent output. The buttons can be used to generate stimuli to fulfill the IOS-UI1 and IOS-UI2 requirements and the LEDs can be used to display the heat levels for the left and right seat, which fulfills the IOS-UI3 requirement.

The *Tamper* and *Key* buttons are used to toggle the passenger detection for the left respectively the right seat (IOS-UI2). The two leftmost LEDs represent the left seat level and the two rightmost LEDs represent the right seat level (Figure 19). The *Up* and *Down* keys on the Joystick is used to adjust the heat level (IOS-UI1). This is only done for *Case 1* when all the SWCs of the demonstrator are mapped to the same node. For *Case 3* when only a *SeatSensor* is mapped to the node, the button *Key* is used to toggle the passenger on seat detection.



Figure 19: Buttons and LEDs on the STM32 board

#### 6.12.2.2.2 Configuration

To allow SWCs to read or write values from or to peripherals on the STM32 board, an *IO hardware abstraction layer* must be generated. This layer is automatically generated by the ArcCore BSW Builder once the necessary BSW modules are configured (DIO, PORT, MCU). Instead of implementing the code manually, the necessary code is generated with the tool. You simply connect the IO ports on the SWCs to the generated IOHwAb channels to read or write values.

The STM32 board has four LEDs and for each LED it is necessary to create a specific port on the SWC that is used to turn the LEDs on or off. The *SeatHeater* SWC used in the demonstrator has two only IO ports (*LeftSeatHeaterIO* and *RightSeatHeaterIO*) and that is simply not enough to address all four LEDs. In order to turn the LEDs on and off according to Table 25: Behavior of the output depending on the input, an additional SWC called *SeatHeaterIOBridge* had to be created which splits the two IO ports on the *SeatHeater* component into four ports. Based on the values written to *LeftSeatHeaterIO* and *RightSeatHeaterIO* ports, the *SeatHeaterIOBridge* SWC toggles the LEDs according to Table 25.

#### 6.12.2.2.3 Implementation

Due to some unknown reason, we were unable to configure the modules to read values from available buttons. Much time was invested to find the problem, but cause could be found. It should

be possible since the same configuration is done to be written to peripherals. In order to read the buttons some manual coding had to be performed which resulted in three functions:

- **void STM\_EVAL\_PBInit(Button\_TypeDef Button)** – Before the buttons can be read they must be configured correctly. The available hardware drivers are used to configure the pins, ports and clocks correctly.
- **uint32\_t STM\_EVAL\_PBGetState(Button\_TypeDef Button);** – This function is used to read the current state of the selected button.
- **uint32\_t STM\_EVAL\_GetHeatLevel();** – Since there are no physical knob available on the STM32 board it had to be simulated. This function is called to read the desired heat level and delivers values from 0 to 2. The value is increased or decreased depending on which Joystick key is pushed.

Source code available in Appendix 6: STM32 button source code.

### 6.13 CAN communication with STM32

Only configuration with Arctic Studio has to be done in order to send CAN-messages with the STM32 board. Once the necessary BSW modules have been configured properly, it is only a matter of connecting the right SWC ports to *System Signals* which are sent over the bus and schedule the transmission and reception manually.

There are seven modules which must be configured in order to activate transmission and reception of CAN-messages on the STM32 board:

1. **Mcu** – The correct clocks must be configured and activated.
2. **EcuC** – The global PDUs must be configured.
3. **Com** – The system signals contained in PDUs must be created and configured.
4. **PduR** – The routing of signals must be correctly performed.
5. **CanIf** – The Can Interface must be correctly configured
6. **Can** – The hardware settings must be configured properly.
7. **Port** – The ports on the board must be activated and configured properly.

The configuration for these modules can be found in Appendix 7: Settings for Arctic Studio modules to enable CAN.

### 6.14 CAN communication with COQOS-tP

Using COQOS-tP to set up CAN communication is no problem down to the communication part of the SL. Although the models can be configured in Artop all the way down to the MCAL there is no support by the COQOS BSW generator to generate the required modules needed to send and receive data via CAN<sup>5</sup>. Since we need to connect multiple ECUs and communicate via CAN the implementation of the necessary CAN modules was something we had to do ourselves.

#### 6.14.1 CAN hardware

When we decided what CAN hardware to use no thorough research was made like in the case of tool vendors. First of all it was not part of the thesis description and secondly we discovered very late that we had to implement the bottom part of the communication stack ourselves thus there was no time

---

<sup>5</sup> Reference: Carsten Krüger, phone call (21/1/2011)

to compare different vendors in order to find the most suitable one. As an attempt to save time we began by having a look at hardware from a vendor called IXXAT which had been used by CrossControl before in their Windows based solutions. At that point it was not known whether or not IXXAT had Linux support for the hardware we had access to but luckily we discovered that you could get hold on a Linux driver on request.

#### 6.14.1.1 IXXAT USB-to-CAN compact

The CAN hardware our CAN implementation is based on is called *USB-to-CAN compact* and is a small and easy-to-use interface which allows us to connect our Linux laptop to a CAN network via USB. The Linux driver was delivered very quickly and it contained an example for every supported hardware interface as well as well-written documentation. IXXAT's Linux driver and API is part of something they call *Embedded Card Interface* (ECI) which is intended as a system extension to allow applications to access different IXXAT interfaces independently of the hardware.

##### 6.14.1.1.1 API Specification

The ECI API is quite large and only the parts directly affecting our implementation are described below.

##### 6.14.1.1.1.1 Data types

These data types are the same for every hardware interface (see Table 28).

Name	Description
<b>ECI_CTRL_CONFIG</b>	Contains configurations for a CAN or LIN controller. For CAN it holds information about baud rate and operation modes such as standard or extended CAN.
<b>ECI_CTRL_HDL</b>	A handle to a controller.
<b>ECI_CTRL_MESSAGE</b>	Contains an identifier and data for a message that can be used when communicating over the network. Can be either a CAN or LIN message.
<b>ECI_HW_INFO</b>	Contains hardware information, for example if it a CAN or LIN controller and what state it is in.
<b>ECI_HW_PARA</b>	Contains hardware configuration parameters such as type of hardware (USB, PCI etc.) and if the hardware should run in polling or interrupt mode.
<b>ECI_RESULT</b>	Return type for all API functions described below. The value is checked in order to discover if a previous API function call returned with an error or not.

Table 28: ECI API data types.

##### 6.14.1.1.1.2 Functions

All functions described below (see Table 29) are prefixed *ECI109*. ECI has already been explained but the number 109 is associated with the hardware we are using, namely the *USB-to-CAN compact* interface.

Name	Description
<b>ECI109_Initialize</b>	Initializes the internal hardware structures; must be the first function called.
<b>ECI109_GetInfo</b>	Gathers and returns hardware information such as controller types and states.
<b>ECI109_CtrlOpen</b>	Opens a controller of the hardware passed as a parameter. The opening procedure comprises initialization of the controller and loading of

	firmware.
<b>ECI109_CtrlStart</b>	Starts the controller communication.
<b>ECI109_CtrlSend</b>	Writes a message to the outgoing message buffer of the controller.
<b>ECI109_CtrlReceive</b>	Reads one or more messages from the incoming message buffer.
<b>ECI109_CtrlStop</b>	Stops the controller communication.
<b>ECI109_CtrlClose</b>	Closes the controller.
<b>ECI109_Release</b>	Releases internal hardware structures.

**Table 29: ECI API functions.**

#### 6.14.1.1.2 Start-up and shut-down

In this section brief descriptions of how the *USB-to-CAN compact* interface internally uses a state machine in order to prepare it for operation and close it down afterwards. There are multiple function calls involved and they have to be called in a particular order for everything to work. *ECI109\_CtrlSend* and *ECI109\_CtrlReceive* are not part of either of these phases and are instead used by the Can module; information about this can be found in the interaction section of the CAN modules.

There are two sets of states in the ECI API; controller and library states (see Table 30). Although there are two sets of states they can be combined into one diagram which is illustrated in Appendix 4: IXXAT ECI state machine.

Set	Name
<b>Controller</b>	<i>ECI_CTRL_UNCONFIGURED</i>
	<i>ECI_CTRL_INITIALIZED</i>
	<i>ECI_CTRL_RUNNING</i>
<b>Library</b>	<i>ECI_UNINITIALIZED</i>
	<i>ECI_INITIALIZED</i>
	<i>ECI_CONFIGURED</i>

**Table 30: USB-to-CAN compact states**

Before the *USB-to-CAN compact* interface is operational a series of states need to be translated. Initially the library sits in the *ECI\_UNINITIALIZED* state. Only one transition is available from this state and it is made by calling *ECI109\_Initialize*. As a result the library is initialized (*ECI\_INITIALIZED*) however the controller remains not configured (*ECI\_CTRL\_UNCONFIGURED*). The next step in the start-up phase is to translate to the *ECI\_INITIALIZED/ECI\_CTRL\_INITIALIZED* state; this is achieved by calling *ECI109\_CtrlOpen*. As far as the library is concerned it is now setup and ready but one step remains for the controller; namely to translate to *ECI\_CTRL\_RUNNING* which will allow it to start communication over the network. This final transition is realized by *ECI109\_CtrlStart*.

Once these transitions have been made successfully CAN messages can now be received and sent by the Can module.

When there is no longer any need to communicate over the network the library and controller can be stopped in a similar manner as the start-up phase; the same states are translated but in the opposite order. If the controller is in the *ECI\_CTRL\_RUNNING* state a call to *ECI109\_CtrlStop* will result in a transition to the *ECI\_CONFIGURED / ECI\_CTRL\_INITIALIZED* state. When sitting in this state the network communication can still easily be made operational without re-initializing anything which is not the case if *ECI109\_CtrlClose* is called since it will trigger a transition to the *ECI\_INITIALIZED /*

*ECI\_CTRL\_UNCONFIGURED* state. The only remaining step required to facilitate a transition back to the initial state of the library is a call to *ECI109\_Release*; unlike any other function it can be called from any state. In other words if there are no plans to stop the controller temporarily and keep the configuration the *ECI109\_Release* function can be called directly when closing-down the network communication instead of performing all the steps mentioned earlier in this section.

### 6.14.2 Implementation of CAN modules

The following CAN modules (see Table 31) are defined in AUTOSAR 3.1 [57]:

Name	Short name	BSW layer	Description
<b>CAN Driver</b>	Can	MCAL	The CAN driver provides, independently from the hardware, services such as transmitting data and notifies upper layers of received data.
<b>CAN Interface</b>	CanIf	ECUAL	The CAN interface makes it possible for upper layers to interface the CAN bus the same way regardless of if the bus is located internally or externally. Since it is located in the ECUAL it also abstracts from the layout of the ECU hardware.
<b>CAN Transceiver Driver</b>	CanTrcv	ECUAL	Driver for external CAN transceiver; provides control of wake-up/sleep events and network diagnostics.
<b>CAN Network Manager</b>	CanNm	SL	When CAN is in interrupt mode the network management module provides bus diagnostics, error recovery, network configuration monitoring etc.
<b>CAN Transport Layer</b>	CanTp	SL	The CAN transport protocol segments data to be transmitted and collects data received on the bus.
<b>CAN State Manager</b>	CanSm	SL	Controlling states of the CAN bus.

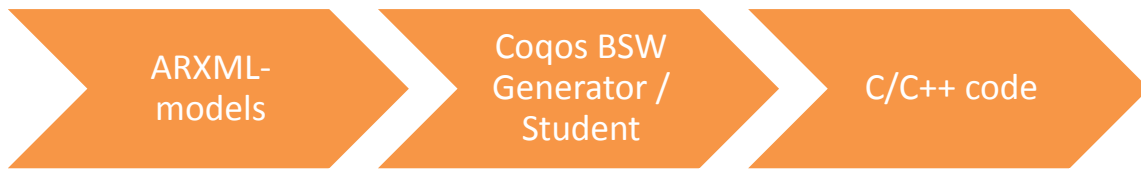
Table 31: AUTOSAR CAN modules

All of these modules are not required in our system and implementing all of them would be a massive task in itself and out of scope for this thesis. For this reason and the fact that COQOS-tP does not provide implementations for any of them, only the modules considered necessary were implemented for our system.

In order to comply with the AUTOSAR standard as much as possible without investing too much time a decision was made to implement the *CAN Driver* (Can) and *CAN Interface* (CanIf) modules only. This allows our system to communicate using the CAN bus; in addition it also allow modules residing in the SL layer to gain access to CAN independently of the CAN hardware and its location. Since we could do without CAN states, bus diagnostics and error recovery the *CAN Network Manager* (CanNm) and *CAN State Manager* (CanSm) modules were excluded. Furthermore, as we are not using any external CAN transceivers nor complex data types, in need of segmentation in order to fit in CAN messages, the *CAN Transceiver Driver* (CanTrcv) and *CAN Transport Layer* (CanTp) modules could be omitted as well.

The Can and CanIf modules were implemented in a similar manner as the modules generated by the COQOS BSW Generator. A decision was made to configure the modules' models in Artop like the rest of the modules used in our system but the step where they are converted into code by COQOS-tP is,

as mentioned earlier, not available at this time. In other words, we had to do the work of the generator and convert the models into code ourselves. By using this approach we can look at it this way: The input and output is structured and named according to the conventions in the AUTOSAR standard no matter if COQOS-tP would have generated the CAN modules or if it was converted manually by hand (see Figure 20).



**Figure 20: Workflow in three steps.**

One could argue that it is redundant to first configure the models and then convert them into code instead of writing the code directly thus eliminating the first two steps.

However if in the future the current tool suite supports CAN module generation, or a switch is made to a tool suite with already existing support, and all modules are configured (step 1) nothing has to be changed in order to have your entire system generated. As a result the whole system can be modeled once and not part by part as generator features become available. In the case where a switch is made to a different tool vendor there could of course be problems if the models contain any vendor specific sections (further details in section 7.3.5.1.1.1). This reasoning obviously applies to other tool suites than COQOS as well. In our situation where modules defined in the standard were excluded, this way of thinking is still valid since those modules could still have models only not converted until there is support for it by a generator thus consequently replacing the manually converted code.

The standard documents of the AUTOSAR models consist of not only what task a particular module shall perform and what it looks like in detail, but also how the file structure (source and header files) shall be organized. Since all CAN modules were not implemented complete compliance was not possible but was followed to the degree possible.

With all this in mind, we proceeded to the implementation phase.

#### **6.14.2.1 CAN Driver**

It is thoroughly specified what the Can module shall do and how. Although it was not implemented exactly according to the specification all the interaction between modules are done the correct way. Most differences between the specification and the actual implementation are because of the other CAN modules than were excluded in the previous phase.

##### **6.14.2.1.1 Module Configuration**

This section will briefly mention the configuration parameters of the Can module and how they are put together into containers (see Table 32). It is based on these configuration parameters the models are created in Artop and later converted into code. For a complete description, see [63].

Name	Description
<b>Can</b>	Contains the entire configuration of a CAN Driver i.e. a CanConfigSet and CanGeneral container.
<b>CanGeneral</b>	Contains parameters related to each CAN Driver component for example time between two API calls to Can_MainFunction_Write and Can_MainFunction_Read.
<b>CanController</b>	Contains the configuration of a CAN controller; for instance CAN controller id and baud rate.
<b>CanHardwareObject</b>	Contains parameters of a CAN hardware object such as CAN id, object handle and object type (transmit/receive).
<b>CanFilterMask</b>	Contains parameters of the CAN filter mask; how CAN identifiers shall be filtered in hardware.
<b>CanConfigSet</b>	Contains multiple configuration sets of CanController and CanHardwareObject for the CAN Driver.

**Table 32: Can module configuration containers.**

#### 6.14.2.1.2 API Specification

In this section data types and functions for the Can module are presented.

##### 6.14.2.1.2.1 Data types

Data types affected by the exclusion of some CAN modules are omitted; the ones that were implemented can be found below (see Table 33).

Name	Description
<b>PduIdType</b>	A PDU identifier can be defined as either 8 or 16 bits long depending on how many PDUs are used by the system.
<b>Can_ConfigType</b>	Contains hardware specific initialization data for the CAN driver and controller configuration structures.
<b>Can_ControllerConfigType</b>	Contains initialization data for one CAN controller. <i>Can_ConfigType</i> may hold several references to this type.
<b>Can_PduType</b>	Contains CAN id, PDU id, DLC (Data Length Code) and data.
<b>Can_IdType</b>	CAN identifier, can be defined as either 16 or 32 bits depending on whether Standard CAN or Extended Can is used.
<b>Can_ReturnType</b>	Return value of the Can module API. <i>CAN_OK</i> if an operation was successful, <i>CAN_NOT_OK</i> an error occurred during an operation or <i>CAN_BUSY</i> if a transmit request could not be executed.

**Table 33: Can module data types.**



#### 6.14.2.1.2.2 Functions

Only the functions necessary to allow communication via CAN were implemented (see Table 34). The syntax of the functions can be found in the Can specification [63].

Name	Description
<b>Can_Init</b>	Initializes the Can module.
<b>Can_InitController</b>	Initializes CAN controller specific settings only. Called once for each controller.
<b>Can_Write</b>	Calls the underlying CAN hardware and attempts to send a Can message.
<b>Can_MainFunction_Write</b>	When the Can module is in polling mode this function periodically checks for confirmations of sent messages (TX confirmations).
<b>Can_MainFunction_Read</b>	When the Can module is in polling mode this function periodically checks for received message indications (RX indications).
<b>Can_DeInit</b>	This function is not defined in the specification. Its purpose is to clean up when the Can module is not needed anymore.

**Table 34: Can module functions.**

#### 6.14.2.1.3 Limitations

Apart from the obvious limitations of the Can module due to the exclusion of other CAN modules there are a couple of other limitations worth mentioning.

- If concurrent calls are made to *Can\_Write* dealing with the same Hardware Transmit Handle (HTH), no check is made to see if it is free or not.
- If *Can\_Write* is busy with a transmit request of a PDU with a lower or equal priority a new request with a higher priority does not preempt the on-going request. Transmit cancellation is not implemented which would have been necessary if the higher priority request were to preempt the lower or equal one.
- *Can\_MainFunction\_Write* is only polling Tx confirmations and not Tx cancellations as well.
- Tx confirmations are defined in the specification as acknowledgements from other ECUs on the network on successful reception [4, p. 39]. Since we don't send acknowledgements but still want to signal upper layer modules our Tx confirmations are simply acknowledging that a Tx PDU was successfully sent by *Can\_Write*. The polling of these confirmations is done by *Can\_MainFunction\_Write*.
- CAN messages are not buffered by the module.
- CAN hardware dependent; the CAN module would have to be partly re-written in case of different hardware.
- No errors are reported to the Development Error Tracer (DET) or Diagnostics Event Manager (DEM) modules.
- Whether or not the correct version of the module is included is not checked.

### 6.14.2.2 CAN Interface

#### 6.14.2.2.1 Module Configuration

As with the Can module a number of configuration containers were implemented for the CanIf module as well in order to be able to call its functions the same way as intended by the specification (see Table 35). For a complete description, see [64].



Name	Description
<b>CanIf</b>	Contains all the necessary configuration containers for the CAN Interface. For example: CanIfControllerConfig and CanIfInitConfiguration.
<b>CanIfPrivateConfiguration</b>	Private configuration for the CAN Interface such as which software filter should be used and if DLC check is supported.
<b>CanIfInitConfiguration</b>	Contains the initialization parameters for the CAN Interface including the CanIfRxPduConfig and CanIfTxPduConfig containers to mention two.
<b>CanIfTxPduConfig</b>	Contains information about a Tx PDU, for example, CAN id, PDU id and the hardware object associated with the PDU in question.
<b>CanIfRxPduConfig</b>	More or less an Rx PDU's counterpart to CanIfTxPduConfig.
<b>CanIfControllerConfig</b>	Contains the configuration of all CAN controllers used by each CAN Driver.
<b>CanIfInitControllerConfig</b>	Contains information about how each CAN Driver is setup.
<b>CanIfDriverConfig</b>	Contains the configuration for each CAN Driver, for example, whether or not transmit confirmation and receive indication are supported.
<b>CanIfInitHohConfig</b>	Is part of CanIfInitConfiguration and contains the configuration containers for all hardware objects.
<b>CanIfHthConfig</b>	Contains the parameters for each HTH. For example a reference a hardware object and an id to the CAN controller owning it.
<b>CanIfHrhConfig</b>	The counterpart to CanIfHthConfig but for a hardware receive handle (HRH).

**Table 35: CanIf module configuration containers.**

#### 6.14.2.2.2 API Specification

In the following section data types and a subset of all functions of the CanIf module are presented.

##### 6.14.2.2.2.1 Data types

In our implementation of the CanIf module not all data types in the specification were used. Below is a table of the ones we needed (see Table 36).

Name	Description
<b>Std_ReturnType</b>	A standard return type which is used by both the RTE and the BSW. The predefined values are E_OK and E_NOT_OK; both are self-explanatory.
<b>PduIdType</b>	A PDU identifier can be defined as either 8 or 16 bits long depending on how many PDUs are used by the system.
<b>PduLengthType</b>	The length of a PDU in bytes. The number of bits depends on the communication system and whether or not segmentation of PDUs is used.
<b>PduInfoType</b>	Stores the information of a given PDU such as the actual data and its length.
<b>Can_PduType</b>	Contains CAN id, PDU id, DLC and data.
<b>Can_IdType</b>	CAN identifier, can be defined as either 16 or 32 bits depending on whether Standard CAN or Extended Can is used.
<b>CanIf_ConfigType</b>	Contains post-build initialization data for the CAN Interface for all CAN Drivers.
<b>CanIf_ControllerConfigType</b>	Is part of CanIf_ConfigType and contains for example which type of wake-up is supported. <i>CONTROLLER</i> means wakeup is performed by the CAN controller, <i>NO_WAKEUP</i> when no wakeup is supported and

Table 36: CanIf module data types.

#### 6.14.2.2.2 Functions

Only a handful of the functions in the specification were implemented but enough to allow communication with other ECUs on the network (see Table 37). The syntax of the functions can be found in the Can specification [64].

Name	Description
<b>CanIf_Init</b>	Initializes internal and external interfaces of the CanIf module. Can controllers are not initialized by this function.
<b>CanIf_InitController</b>	Sets up buffers for Tx and Rx PDUs for a specific CAN controller.
<b>CanIf_Transmit</b>	Initiates a request to send a PDU.
<b>CanIf_TxConfirmation</b>	The CAN Driver calls this function when a PDU has been successfully sent.
<b>CanIf_RxIndication</b>	The CAN Driver calls this function when a PDU has been successfully received; the PDU is then routed to upper layers.

Table 37: CanIf module functions.

#### 6.14.2.2.3 Limitations

Notable limitations of the CanIf module:

- CAN messages that could not be sent by the Can module are not buffered in the CAN Interface; instead they are discarded.
- Transmit cancellation is not supported.
- Received CAN messages are not buffered by the module.
- The PduR module is the only CanIf user notified when a PDU has been transmitted or received.
- No multi Can Driver support.
- No errors are reported to the DET or DEM modules.
- Whether or not the correct version of the module is included is not checked.

If one is to strictly comply with the specification there obviously a lot more limitations; but given the time we had to implement this there was no other option.

### 6.14.2.3 Interaction

The interaction between BSW modules in the lower part of the communication stack is mainly between Can, CanIf and PduR; if the rest of the CAN modules are implemented those are of course included as well. To more easily visualize how the CAN modules we implemented interact with the rest of the communication stack we will in this section describe a set of typical scenarios.

#### 6.14.2.3.1 Transmit request

When the RTE decides data needs to be sent to a SWC located on a different ECU it uses the communication stack. The data is sent via the Com and PduR modules before it finally arrives at the CanIf module. What happens above the PduR module is not relevant in this section thus any further details are omitted.

Typically in this scenario one module is seen as the user, in our case it is the PduR module since it is the one calling CanIf requesting to send a PDU. PduR uses the function *CanIf\_Transmit* provided by

CanIf to notify what PDU it wants to transmit. If CanIf can find the Tx PDU id in one of the *CanIfTxPduConfig* containers it forwards the request to the Can module using *Can\_Write*; in case no configuration container is found an error is returned. When Can receives a call from CanIf, and it has been initialized properly, it goes ahead and transmits a CAN message by calling *ECI109\_CtrlSend* of the CAN hardware. If the module was not initialized before an attempt to transmit, or an error occurred during transmission, *CAN\_NOT\_OK* is returned.

An illustration can be found in Appendix 3: CAN sequence diagrams.

#### 6.14.2.3.2 Transmit confirmation

Our implementation is using polling mode when handling transmit confirmations. To call our main function *Can\_MainFunction\_Write* which performs the task of forwarding transmit confirmations we use the *BSW Scheduler module* (SchM) [65]. SchM processes main functions of BSW modules such as Can and Com. It can be configured to trigger events sporadically and cyclically, the latter is what we are using.

As previously mentioned the whole procedure of processing transmit confirmation is started by the SchM when it calls *Can\_MainFunction\_Write*. All Can does is check its list of Tx confirmations and forwards the ones it finds to CanIf using *CanIf\_TxConfirmation*. When the list has been traversed all elements are deleted in order to prevent confirmations from being sent multiple times. If CanIf is able to find the PDU related to the Tx confirmation in the *CanIfTxPduConfig* container and a user is specified for that particular PDU the user is notified. As in the case for the transmit request scenario, our only user is PduR and it provides a function called *PduR\_TxConfirmation* for this type of notification.

An illustration can be found in Appendix 3: CAN sequence diagrams.

#### 6.14.2.3.3 Receive indication

Handling of receive indications are set to polling mode just like for transmit confirmations hence triggering of this scenario is similar to the one described in the previous scenario only this time the main function being called is *Can\_MainFunction\_Read*. *Can\_MainFunction\_Read* tries to read from the ECI109 CAN hardware by calling *ECI109\_CtrlReceive*. If one or more CAN message has been received CanIf is notified once for each message via a call to *CanIf\_RxIndication*. According to the specification whether or not CanIf performs software filtering and DLC check is optional. Software filtering is an approach to discard received messages not within the defined range of CAN ids for Rx PDUs. A DLC check is performed in order to discard CAN messages with a DLC smaller than the expected one which was statically set. Only the DLC check is supported by our implementation. Once it has been established that a correct CAN message has been received CanIf notifies the user (PduR) by a call to *PduR\_RxIndication*.

An illustration can be found in Appendix 3: CAN sequence diagrams.

## 6.15 Debugging

From experience from before and during our thesis, we know that debugging is a crucial part of the application development process.

### 6.15.1 Debug precompiled applications in Atollic TrueSTUDIO

One major problem when using Arctic Studio is that there are no native debugging possibilities. You develop your application, compile it and then use some kind of flash programmer that is specific for the hardware code loader in use. This workflow is undesirable and time consuming. It is also quite difficult to find errors and bugs in the code.

What we need is an application that can handle precompiled solutions (.ELF) and debug them. We tried to use Keil UVision 4 from ST and ARM Workbench from IAR without any success. For the record, none of them are based on Eclipse.

After the struggle, we tried to see if there could be other open source options for us to choose. There are plugins to enable various JTAG debuggers.

We came across Atollic TrueSTUDIO[66] and is just like Arctic Studio based on the Eclipse framework. This is a major advantage since this makes it possible for us to compile the application in Arctic Studio and then debug it using TrueSTUDIO.

The debug workflow is not perfect, but better than the first tries when we tried to share the same workspace between the two Eclipse applications, which is not possible to do, but you are allowed to import and link the files from Arctic Studio's private workspace to Atollic TrueSTUDIO's workspace.

#### 6.15.1.1 Steps to debug the software

This section could be seen as a guide for others who might try to accomplish the same thing as us. The solution should be applicable to setups similar to ours.

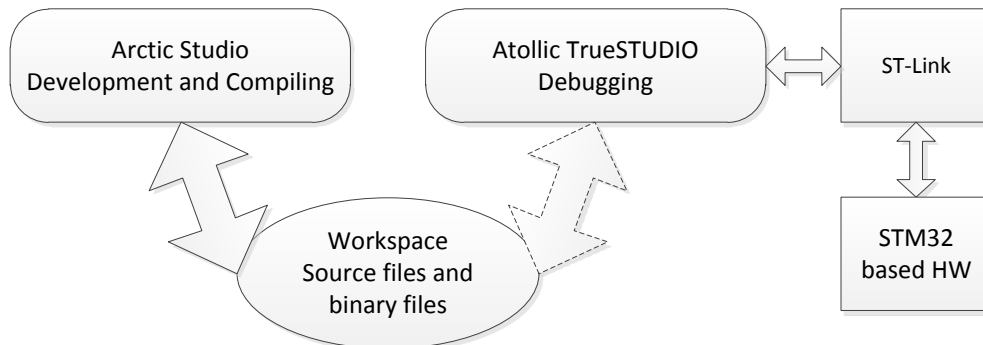


Figure 21: Relationship between Arctic Studio and Atollic TrueSTUDIO

Our setup is visualized in Figure 21. As mentioned before we use Arctic Studio to create components that we later will implement. Once the configuration of the components is done and the behavior of each runnable is written, it is time to compile the solution into an .ELF-file with Arctic Studio.

1. Start Atollic TrueSTUDIO and select a workspace. It is possible to choose the same workspace as Arctic Studio, but is not recommended.
2. Right click in the Project Explorer and select Import.
3. Under the General tab, select to import an existing project into the workspace.
4. Browse to the root folder of the Arctic Core projet you wish to debug.
5. Do not copy the project into the workspace. Then press finish.
6. Now select Run and open the Debug Configurations window. This is where you will configure your debug session

7. Create a new Launch Configuration
  - a. Give the configuration a name, preferably the same name as the imported project.
  - b. Browse and select the imported project.
  - c. Browse for the compiled ELF application you wish to debug. In our case, the file can be found in: `/Arctic Studio/workspace/<project_name>/obj_<used_board>/`.
8. Depending on your card manufacturer, it might be necessary to change the *GDB Server Command Line Options* to SWD or JTAG under the Debugger tab.
9. Nothing needs to be changed in the Startup, Source or Common tabs.
10. Press Apply and then Debug.

This method is repeated for each imported project that is going to be debugged. When you wish to debug your project, you simply run the correct Debug Configuration.

**Note.** We have been using v1.4.0 of Atollic TrueSTUDIO and it appears that this setup does not work when using the latest version (v2.0.1) since it does not allow the user to ignore the settings concerning the used hardware; we have already made this choice in Arctic Studio and we can therefore ignore it with our setup.

### 6.15.2 Linux

Typically an application is implemented and debugged using an IDE but since this was not possible directly with what Open Synergy provided a different approach was needed. According to Open Synergy their customers have customized solutions for these features but this was not investigated much further due to the fact that it seemed like a major task in itself to do a similar customization for our demonstrator. Instead the demonstrator prints debugging information to the Linux terminal thus providing the necessary information to be able to tell where and when something goes wrong if that is the case. However there was no substitute for the ability to step through the code which is a common feature in every IDE.

When the Can and CanIf modules were created it was possible to isolate them from the rest of the system during the development phase which allowed us to easily debug them using Eclipse. Although it is believed that the same thing could have been done for the entire system it was, as mentioned earlier, considered an overwhelming task that would only have been investigated further if time had allowed it or due to bugs that are difficult to trace.

The I/O simulator developed for Linux is completely decoupled from the demonstrator so the complexity of it did not affect the possibility to have debugging capabilities without a lot of extra work. Qt Creator provided all the necessary functionality in order to implement and debug the simulator efficiently.

## 7 Evaluation

This section is based on the observations gathered while creating the demonstrator in chapter 6 and the formulation of the questions of this thesis.

### 7.1 Hardware independence

Earlier ECU solutions have usually been tightly coupled with a specific hardware, making it hard or impossible to unite HW and SW from two different manufacturers. There is a high degree of

*dependency* between the HW and the SW. The main idea behind AUTOSAR is to loosen this dependency, so it would be possible to combine HW and SW from two different dealers.

During our research we have found that this dependency surely is much lower than before, but you are still not totally free when it comes to select the AUTOSAR core from different vendors.

### 7.1.1 Compilers

We encountered this problem but it shall be noted that this matter has not been fully researched.

One thing that is important, and might not be obvious, is that you are limited to what compiler you are able to use when selecting a certain AUTOSAR core. The OS API that is specified in the standard [67], states that the function calls are defined according to the C89. C99 allows dynamic memory allocation during run-time, which is not commonly used in automotive applications. Yet still, what it looks like, free to implement according to C99.

The core from ArcCore needs to be compiled with a compiler that supports C99 and GCC extensions. They have chosen to do so, but have plans to rebuild their core so this you are free to choose which compiler you wish to work with.[68] Today they deliver a preconfigured compiler for a couple of architectures, such as ARM which the STM32 board is based on.

We have used an evaluation board from IAR, and their compiler doesn't support C99 [69], however, according to their website they have some support, but only a fraction of the whole C99 standard..

### 7.1.2 Memory

One factor that limits the freedom when selecting the hardware platform is the internal memory of the chip. When we studied the vendors' homepages to find information about their AUTOSAR solutions, we never found a single statement concerning the required minimum size of the internal memory. This is quite troublesome since you might have a processor type that is supported, but the current version of the chip doesn't have enough internal memory, a problem we had to encounter.

Since we have not been able to get the necessary information from more than just one company, due to the lack of interest from other companies, it is not safe to draw conclusions about the minimum memory at this point. What we can say for certain is that memory is a limitation, but to what degree still remains unanswered.

	Not recommended	Recommended
<b>Flash</b>	128K	256K
<b>RAM</b>	20K	64K

Table 38: ArcCore's recommendations

What we can see in Table 38 is the information we got provided from the company ArcCore after we posted them a concrete solution of a development board we had planned to use during our tests. Their first reflection was that the internal memory in our board would not be sufficient, which later turned out to be an accurate guess. There was a risk it would not be possible to use their AUTOSAR core, at least not without optimization.

We have tried to get in touch with other companies, but only three of the ones we selected to contact, responded to our emails and form fill outs.

There is a risk that the memory in the microcontroller is not sufficient to hold an entire AUTOSAR solution.

### 7.1.3 Microcontroller

As written earlier, applications written with the AUTOSAR standard in mind will be hardware independent. As we travel down the architecture stack for AUTOSAR, from the Application layer all the way down to the bottom of the BSW layer, the hardware dependence must start somewhere along the way.

The lowest layer in the BSW layer is the so called Microcontroller Abstraction Layer; this layer is written and adapted to support different hardware, this is where the highest degree of hardware dependency can be found.

What we have noticed during our research is that it can be quite hard to get the necessary information about the supported microcontrollers. We initially thought that the companies would have support for the most common microcontrollers, but in reality, it is a mixed bag. Some of them offer great lists over the supported microcontrollers, e.g. Vector[70]. Others like Elektrobit, at least give a hint of which microcontrollers their solution supports [71]. We cannot find a good reason why most of the companies we have examined don't present a complete list of supported microcontrollers. These withhold of information only makes it tougher to select an AUTOSAR implementation that suits a company. It might be the case that the business market for AUTOSAR is still so fresh, that the companies that offer solutions which to have all doors open for customers. Revealing information about the supported microcontrollers too early, might lead to a loss of customers, this is however just a speculation from our side.

***This answers Q2.6 "How important is it that the development tools support the target hardware?"***

Our initial thought was that this would not be a big problem, but we were wrong and thanks to some of the companies we have talked with, we now know that dependencies caused by the microcontrollers, is a cornerstone when selecting the right AUTOSAR core and tool.

### 7.1.4 Development platform

Depending on the solution you are forced to choose a development platform that suits the target platform. The development platform dependability is not related to the AUTOSAR standard itself, the dependence is connected to solutions that are based upon the AUTOSAR standard. One of the vendors we came across during our research was OpenSynergy based in Berlin, Germany, and their AUTOSAR solution really stands out from the crowd.

Instead of having microcontrollers and hardware at the bottom of the architecture, they have built a wrapper that makes it possible to run the entire AUTOSAR architecture on top of a Linux kernel. This however results in an additional dependency, which we realized after we had to contact the company.

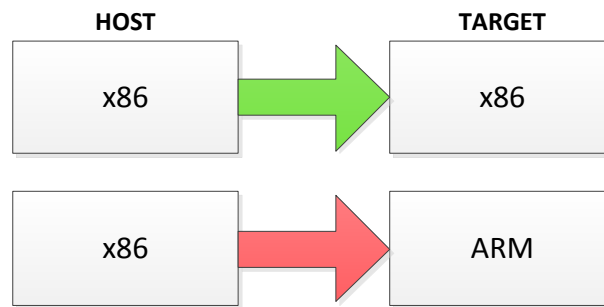


Figure 22: Development platform and target

The dependency is related to the problem with cross-compiling, where the host platform and target platform has different type of hardware (Figure 22). OpenSynergy's solution suffers from this problem; the target platform must be the same as the host/development platform<sup>6</sup>. This significantly limits the products that could be used with OpenSynergy, especially since most embedded computers have limited resources and a wide variety of different architecture. However, depending on the system you are working with, this might not be a problem, as long as the system will contain a car-pc rather than a simple ECU.

## 7.2 Maturity of the tools

The initial start for AUTOSAR was back in 2003 and the first revision (R1.0) of the standard was released 2005. Since then, five larger revisions of the standard has been released and the latest one (R4.0) was released December 2009. With each new revision of the standard, new modules are added and new documents are added or old ones updated.

Many of the documents are then used by the tools vendors in order to create tools that are adjusted to handle the creation of AUTOSAR systems.

The AUTOSAR market is relatively new and small; therefore not so many tools have yet been released.

### 7.2.1 ArcCore

One of the two tools we have been using during our thesis is the Arctic Studio from ArcCore. They have been developing the IDE for some years and in this section we will present some points how the tool could be improved, in order to raise the maturity of the tool.

These are the points:

- **Improvement of testing before releases** – We are aware that it is tricky to test software with all combinations an end user might do. We found minor problems right after new releases that was not too hard to find and discovered during normal cases. We are forgiving since we used their latest releases directly after they are released  
A solution for this issue could be to introduce Use Cases (if that is not already done). The Use Cases should be designed to expose the available modules for normal user interaction and input. One possible solution is to create configuration files for the modules along with tables of the expected generated output. For example, how many files should be generated and

<sup>6</sup> Email conversation with Carsten Krüger



what should they contain. This problem could unfortunately lead to a lack of faith in the software.

- **Generated code and functions**– Most of the functions are very well documented with explanation about the input and output along with an explanation how the function works internally. But there are also many files which we have been looking at that could be improved. If the vendor goes through the code and ensure that all comments are in English and add more comments about the input/output of the functions, the maturity of the tool would be raised by several levels.
- **Incomplete examples** – There are a couple of examples that can be used to learn about some of the parts of the software. However, these are created with a combination of the tool and manual coding. It would be better if the examples was only made with the intended tools and follow the complete tool chain. A fictional project which involves a usual automotive functionality could be used to create a more complete example.
- **Files used for generation** – It is not totally clear which files that are actually used when building the project for the target. There were numerous times when we updated files we believed were used. We have discussed this problem with ArcCore and they are aware of the problem and are not happy how the inclusions of files are performed today and are probably undergo changes in the future.
- **Validation** – The validation is quite good and it is not to tricky to understand what the problems are in the files, yet there is one annoying matter related to this. The software does not only validate the file you are currently working with. All the other .arxml-files are also under the inspection glass and it is frustrating to see errors from other files that you know contain errors or are not working with. One workaround for this is to create filters, but they are not saved once the program is closed, at least not for us.
- **File comparison not found**– This point should be interpreted as a note rather than something else. According to the product sheet for XXX Builder, there should be possible to compare AUTOSAR files and show differences between these, something we have been missing, but we have been unable to find this feature. This was found quite late in the project so we have not had time to ask where it can be found in the tool.

### 7.2.2 COQOS-tP

Overall COQOS-tP gave a good impression in regard to workflow and maturity but there were however a few improvements that could improve the user experience even further:

- **Automated generation and build phases** – Using the generation and build system provided with the example project bundled with COQOS-tP a lot of manual editing was required for every new component that was added. Not only is it a lot more time consuming to get the system compiled and ready to run it also increases the risk of breaking the system when plenty of editing is required in multiple files. The latter happened many of times which resulted in extensive error tracing; this would have been avoided if manual editing was not required.

It would have been very helpful to have an IDE like Arctic Studio that would allow you to not only model the system (which was done in Artop) but also have generate and build support

directly built-in. As soon as for example a new component was added the IDE would automatically add it to the models to be generated and built.

- **Improved generator reliability** – Although the RTE generator and BSW generator shall have praise for their detailed error outputs sometimes the auto generated code did not compile even though the generation phase was successful. If no errors are reported from the previous step one would like to rule out that any errors introduced in the current phase are results of something done wrong in earlier steps. One particular error received when generating our system was also present in the provided example but was never noticed since in that special case it did not affect the system.

***This answers Q2.1 “How does the maturity of the tools look like, are they established?”***

The tools used during the thesis are at different levels of maturity but can be even more mature if the mentioned issues in section 7.2 are taken care of. The standard is growing rapidly and it is not surprisingly that smaller companies are having a hard time keeping up with the standard. Larger companies with more resources should be able to stay in phase with newer revisions of the standard.

### **7.3 Possibility to Share AUTOSAR files within and between tools**

One of the main benefits with standardization is the possibility to share solutions between each other. In this section we investigate how easy it is to share common files between different tools in practice and present our findings.

#### **7.3.1 What do you wish to share?**

The first question we must ask ourselves is what type of information we wish to share between each other.

##### **7.3.1.1 Software components**

AUTOSAR is pushing hard to make the SWCs reusable, thanks to the decoupling from the hardware. This opens the possibility for new market areas, such as the “software-of-the-shelf” market. You are definitely going to share SWCs between different tools or within one tool suite.

##### **7.3.1.2 ECU Extract**

The second thing is ECU extracts. These contain information about which SWCs should be bundled together on a single ECU. It could be interesting to share these files, not only within the same tool, but also between two different tools. A case could for example be that you have your team of developers and some of them are working on the implementation of the SWCs. One of the members then decides how these should be bundled together and mapped to different nodes. These extracts can then be delivered to the guys who are configuring the hardware.

##### **7.3.1.3 Basic Software configuration**

The last thing that could be interesting to share between users is the configuration of the BSW modules, at least the ones that are not directly connected with the hardware. This could for example be the COM or OS module. There are two cases when this could be useful:

1. The company is using only using one tool and wishes to share the configuration within the same tool.
2. Two different tools are used and one of the team members configures the modules for the other team members in order to save some time. A possible reason why the company may use two different tools is that they have different hardware that requires different tools.

### **7.3.2 Creating the test files**

The design of the demonstrator system, which can be found in section 6, is used as a base for the creation of the necessary files. The files used to test the interoperability and sharing between tools was created by using Artop and gedit.

#### **7.3.2.1 Artop**

With the delivery of COQOS-tP we also received a standalone version of Artop, presented in section 4.2, which can be used to create .arxml-files which are used to describe SWCs, Interfaces etc. In this case, we used it to create the description of the SWCs, Interfaces used by the SWCs and data-types used by the Interfaces. For each SWC we created a separate file and the available Interfaces and Data Types were placed together into a single file. We followed the design of the system when we created the necessary files.

We could have used Arctic Studio to create the files since it also uses Artop framework, however, the tool itself has limitations and does not support the whole standard, which Artop does. Therefore, we only used Artop to create the files since it has no limitations and supports the whole R3.1 version of the standard.

The descriptions files for the SWCs, Interfaces and Data Types can be found in Appendix 1: Demonstrator ARXML models.

#### **7.3.2.2 Gedit**

The implementation of each SWC was done using gedit. We choose to use a simple editor instead of a full IDE since the files would not involve so much code. Since we knew how the function header would look like and which header files needed to be included, it was a fairly easy task. It was necessary to use the COQOS-tP generator to get the header files for each SWC, in order to know the correct names of the function calls available for the SWCs. Since the format of these names is standardized, it is no problem to use COQOS-tP to get the correct names, since all generators should generate the same names.

### **7.3.3 Sharing constrains**

As we have stated several times in this paper, reuse and share of SWCs is one of the key factors and benefits with AUTOSAR. Therefore it is quite natural that we wish to examine how we could share SWCs between our tools, which are quite different.

#### **7.3.3.1 SWC Compositions and ECU extract**

We discovered that COQOS-tP has a requirement that all of the SWCs used in the system must be placed in a composition. As mentioned in section 2.5.4, a composition is created by combining smaller SWCs into larger and more complex compositions. These compositions are then placed in the ECU extract in order to generate the code for COQOS-tP. A consequence for this setup is that all involved tools that which to use the ECU extract must support compositions in order to reuse ECU extracts designed for COQOS-tP.

#### 7.3.3.1.1 Compositions not supported in Arctic Studio

As the section name implies, we were not able to reuse the ECU extract created for COQOS-tP since Arctic Studio lacks the support for compositions. So we ended up on a dead end with these tools. There is support to import ECU extracts according to Michael at ArcCore, but this is something we have not tested.

If we had created an ECU extract with Arctic Studio and tried to use it with COQOS-tP, we would have faced the same problem (just in the other direction). Since we cannot create compositions with Arctic Studio and COQOS-tP requires that, we are unable to share these files between the tools, it's a dead end.

We should note that in the latest release of Arctic Studio, which we have not been working with, there appears to be support for compositions when creating the ECU extract. This is something we have not investigated at all since it was released to close the end of the project.

#### 7.3.3.2 *Multiple instantiation of SWC*

One of our criteria for the system is that some of the components should be instantiated several times, in order to test reusability of SWCs and write code which is general rather than specific. In our system, we chose the SWC which handles the seat sensor to be our perfect test subject.

##### 7.3.3.2.1 Not supported in Arctic Studio

With the version of Arctic Studio we have been working with during the thesis, it is sadly not possible to instantiate SWCs multiple times in a system. It is only possible to instantiate one of the available SWC from the library, once. When we created the demonstrator system, we were quite sure that this was possible to do.

##### 7.3.3.2.1.1 *Solution and changes*

In order to make the best out of the situation, the solution to this lack of support was to manually recreate the SWC two times. We simply copied the original version of the SWC and created two different versions of it, one for the left sensor and one for the right. Since the original code and the component were written with the support of multiple instantiation in mind, we had to make some changes:

- **IFDEF unnecessary** – Due to a problem with the generation of code which handles the digital input from the STM32 board, our initial plan was to use IFDEF to distinguish the node used to run the application. This was however not necessary since we had to recreate the files and make them specific for the STM32 board anyway.
- **Update the SWC file** – One of the larger changes we had to do was to update the copied .arxml-file which describes the SWC to match our system description. The whole file must not be updated, only the fields which cannot be named the same. For example, the runnables associated and the name of the component had to be updated.
- **Changed the included files** – Since the name of the component had to be changed, the included header files needed to be updated. The generated header files are named after the component and the generated information in these header files are specific for that component, which makes it impossible to use the same header file for both components.

In our case it did not take so much time for us to manually update the files so they could be reused, but still it consumed some of our time which we did not planned to lose. We only had to work with one component and create two additional ones from it, but imagine how much time it would take to fix this problem in a very large system which heavily relies on the reusability of components.

### 7.3.4 Sharing of Software Components

There are two files connected which each SWC: an .arxml-file which describes the component and a c-file containing the implementation of the component. We have found that these files can be reused in different levels depending on the situation and case. In this section we will summarize what we could reuse from each SWC and what modifications we had to do in order to run the code on both architectures.

#### 7.3.4.1 *SeatSensor* – Large modifications required

As we already have mentioned in previous sections, SeatSensor had to go through some larger modifications in order to use it on the STM32 board and Arctic Studio. The original files which are created according to the demonstrator system design, can run on the laptop with COQOS-tP, but since we had to copy these files and modify them to run on the STM32 board, only a small fraction of the original files are actually reused.

- **Component description** – Copied and modified into two new components.
- **Implementation** - Copied and modified for each new component.

These changes and modifications are unnecessary if only COQOS-tP is used thanks to its support for multiple instantiations of SWCs. When Arctic Studio also have this support, it should be possible to use both files without any problems.

#### 7.3.4.2 *HeatRegulator* – Small modifications required

This component is only used one time in the system, so we would not suffer from the same problem as the SeatSensor. We could completely reuse the component description files, and it would also be possible to reuse the implementation without any modifications. Due to unknown problems, we had problems to generate code with Arctic Studio to handle the Digital Input on the STM32 board, such as buttons and other peripherals. We have successfully been able to write values to peripherals, but not read values from them. To avoid this problem we had to use the drivers for the board and make calls directly to the hardware via the drivers for the STM32 board. This goes against the standard, all communication should go through ports and only lower layers are allowed to make calls to the hardware. We took the decision to ignore this since we know it should be possible to only use the tool to create the necessary hardware connections. To simulate a sensor only capable to deliver three different heat levels, an additional function was created which is used to read the heat level.

Since the calls to the hardware are specific for the STM32 board, we had to distinguish which type of node the implementation would run on. This was solved using the preprocess directive IFDEF in order to include the necessary header files and use the functions to read values instead of reading from the components ports.

- **Component description** – Nothing has to be done, the component can be reused by both tools without any need of modification.

- **Implementation** – Is possible to reuse on both systems if the reading of digital input would have been possible on the STM32 board.

#### 7.3.4.3 *SeatHeater – Small modifications required*

Just like the HeatRegulator component, this component is only used once in the system and therefore we could reuse the entire component description without any problem. However, we had to make changes in the implementation to run the code on both nodes. The problem appears on the laptop where values written on the components ports, is never received by our graphical interface. We created an additional SWC which would handle the communications with the graphical interface through sockets, and connected this new component with the IO ports on SeatHeater. This new components never receives values from the SeatHeater and we have not been able to discover the reason. We therefore made a similar solution which can be found in the HeatReulator implementation when we used IFDEF preprocess directives to run code specific for the laptop. The same reasoning can be applied there, we know that this is not allowed but it is possible to make it work according to the standard when all communications goes through the ports.

- **Component description** – Nothing has to be done, the component can be reused by both tools without any need of modification.
- **Implementation** – Is possible to reuse on both systems if the written values could have been read on the laptop.

#### 7.3.4.4 *SeatHeatingController – No modifications required*

This component was best suited for reusability and sharing between the two tools. It does not rely on the hardware since that is taken care by other components in the system. We could reuse both the component description and implementation between the tools without any modifications to the files.

- **Component description** – Nothing has to be done, the component can be reused by both tools without any need of modification.
- **Implementation** – Nothing has to be done, the implementation can run on both nodes without any modification.

With this component, we revealed that the component itself is decoupled from the hardware and we can use the same implementation regardless of the hardware used, just as the standard states.

### 7.3.5 **Sharing of Basic Software Configuration**

In this section we describe what we encountered while we tested to share the configuration of the BSW modules between the Arctic Studio and COQOS-tP.

#### 7.3.5.1.1 *Between tools*

Since .arxml-files are standardized is should be possible to share the configuration of the BSW between different tools. We discovered that it is possible be extract the information which are stored in these files, but it order to do that, “cut and paste” and “search and replace” must be frequently used, and that technique is in this case very ineffective and prone for errors.

##### 7.3.5.1.1.1 *Vendor specific solutions - SDGs*

One of the reasons why it is inflexible to share the configuration of the BSW modules between tools is because it exists so called *vendor specific solutions* in the .arxml-files they produce.

One of the reasons why it is problematic to share the .arxml-files between different tools is so called *vendor specific solutions*. It is possible to add Special Data Groups (SDGs) tags in the .arxml-files, within these tags; the tool vendor can place data which is specific for the tool [72]. This could for example be information about which version was used to create the file, file output path etc. There are no rules regarding how the data should be formatted within these tags, and that makes it problematic to share these files between different tools.

#### *7.3.5.1.1.2 Not commonly done*

When we proposed our thought for the people at ArcCore, we were told that this sharing between tools is not done in practice, and mentions that one of the reasons is vendor specific solutions. They have tried to keep the numbers low in their tool (Arctic Studio), but to make it practical; some specific solutions must be used.

#### *7.3.5.1.2 Within the same tool*

In this section a description is presented of how sharing basic software module configurations within the same tool works.

##### *7.3.5.1.2.1 Arctic Studio*

There are no problems to reuse previously made configuration of the modules, thanks to the possibility to import modules from other .arxml-files within the tool. All the settings for the BSW modules is bundled together in a single file and it would be tricky to extract the wanted settings without the import support.

##### *7.3.5.1.2.2 COQOS-tP*

Instead of gather all the settings and configuration of the BSW modules into a single file, each module has its own file. There are no import options like in Arctic Studio, but thanks to the separation of the modules, you simply can copy the files you wish to reuse and place them in the new project and they will appear in the tool. Once the files has been copied, there is some work with includes paths before the system can be generated.

***This answers Q2.3 “Is it possible to share and reuse configurations between tools from different vendors?” and Q2.4 “Is it possible to share and reuse configurations between tools from the same vendor?”***

Even though the description format is standardized, it is problematic to share configuration of BSW modules between tools from different vendors due to the use of SDGs. If only tools from one vendor are used, it is much easier to share and reuse the configuration.

***This answers Q2.2 “How is the workflow between tools from different vendors, encouraged or not recommended?” and Q2.5 “Development tool toolchain, is complete better than divided?”***

It should be possible to use multiple tools from different tool vendors and achieve a seamless workflow, as long as the involved tools support the same amount of the standard. If the gaps are too large it can be quite troublesome since one of the tools might produce output which is not supported by the other tools. Therefore, it is recommended to use tools which are planned to work together, either as a complete tool chain or collaboration between vendors.

## 7.4 Trade-offs in practice

Possible trade-offs that may arise when introducing AUTOSAR has been discussed in (section 2.7) and in this section we will briefly present the impressions received after realizing the demonstrator.

### 7.4.1 Memory

We created a system which turns on and off one of the LEDs on the development board and implemented it according to the standard using Arctic Studio and compared it with a pre-made example from IAR Workbench, which also turns the LEDs off and on. The size of the compiled files were then compared. See Table 39 for a summary of the sizes.

	SimpleLedWriter (Arctic Studio)	IOToggle (IAR Workbench)	Factor
<b>Data (kB)</b>	2,4	0,074	32
<b>ROM (kB)</b>	40,8	1,32	31
<b>RAM (kB)</b>	23,9	1,02	23

Table 39: Summary of the built sizes from Arctic Studio and IAR Workbench

There is a large increase of memory usage for the *SimpleLedWriter* compared to the *IOToggle*. Approximately 30 times more memory is required for the same functionality. The largest increase should come from the lower layers of the architecture: RTE and BSW. Thus, simpler applications might not be suitable to use with AUTOSAR. As concluded in [73].

**NOTE:** It shall be noted that the same code is NOT used by both tools and NOT complied with the same compiler. This comparison is merely done to see at what scale the differences are. More research is required to determine a more accurate analysis.

### 7.4.2 Execution Overhead

Instead of measuring the time it takes to send a message out on the CAN-bus, our method was to count how many lines of code it takes before the message is actually sent out on the bus. All necessary initializations are already done and the message has the same properties on both setups. The counting starts from the first function which initiates a transmission phase and ends when the message is sent.

Project (Tool)	Nr of Lines
<b>CAN Example (IAR Workbench)</b>	≈ 50
<b>CAN Transmission (Arctic Studio)</b>	≈ 250

Table 40: Number of lines it takes to send a CAN-message



As we can see in Table 40, it requires approximately five times more lines of code in order to send a message on the CAN-bus with an AUTOSAR project, compared with a non-AUTOSAR project. The consequence is an increased execution overhead in the AUTOSAR project. The main differences between the two projects are the number of layers between the application and the hardware. The application created with Arctic Studio has to travel through many layers and functions before any transmissions since it is not allowed to access the hardware directly. The application created with IAR Workbench is allowed to make a direct call to the hardware, which results in a much lower count of code lines.

***This answers Q1.5 “Has the development of the standard introduced any trade-offs?” and Q3.4 “Does a migration affect the requirements on the hardware?”***

The memory usage and execution time is affected negatively by the introduction of the standard. This issue must be taken into account when hardware is to be selected and solutions migrated.

## 8 Lessons learned and Reflections

In this section lessons learned throughout the thesis is presented along with our reflections afterwards.

### 8.1 Lesson 1: The Workflow could be smoother

The first impression one got after reading up on the AUTOSAR standard was that standardized files such as the arxml-files could be shared between tool suites and not only for a specific vendor. This was however not true in practice since it is heavily dependent on how much of the standard a particular tool suite supports. So called vendor-specific solutions can (see section 7.3.5.1.1.1) also be used within an arxml-file which will prevent other vendors from parsing the file even further. As a result the intended workflow established from the beginning somewhat fell apart since a lot of manual work was required in order to reuse the models of the system in both Arctic Studio and COQOS-tP.

#### 8.1.1 Design system topology at an early stage

Even though an important part of AUTOSAR is about the ability to easily reconfigure the system in order to change the location of SWCs on different ECUs our experience tells us it is not as easy in practice as it seems in theory. If multiple SWCs located on the same ECU are spread out on multiple ECUs it is not enough to change the ECU extract; for example signals and PDUs have to be created which is not automated at least by the tools used in this thesis. If one is working on a system and the placement of SWCs is likely to be distributed on several ECUs we believe the importance of spending a thought or two on the topology of the system early in the process increases along with the number of ECUs of the system.

***This answers Q3.5 “Are there any major pitfalls while migrating? If so, how can they be avoided?”***

Distributing SWCs on multiple ECUs is not as straight forward as the first impression leads one to believe. A lot of unexpected work may be required in later stages of the development. In order to avoid or at least minimize the impact of this it is recommended to early on think of how to split up the migrated system’s functionality into SWCs as well as the desired topology. With this in mind it is easier to plan ahead and avoid unpleasant surprises.

### 8.2 Lesson 2: The Learning curve is steep

To learn something new, you need to put in some effort of varying size, depending on the task. AUTOSAR is no exception; you are facing with a whole new standard which documents that made of thousands of pages. Already it has reached revision 4 and the tool vendors and possibly the users are struggling to handle and adapt to the new requirements and added functionality.

#### 8.2.1 Where to start?

Firstly we are going to give you a presentation about the road of information we have been travelling on and then give some tips and hints on what we consider to be a good reading start.

### **8.2.1.1 Literature**

There are currently no published books that are dedicated to AUTOSAR, except for a few written in German. You might find some automotive books that have at least one small section about AUTOSAR, but no one containing all parts of AUTOSAR.

The number one source for our first encounter with AUTOSAR is the official website for AUTOSAR <[www.autosar.org](http://www.autosar.org)> where all documentation about the standard can be found. They have also a couple of pages that summarize frequently asked questions and highlights some of the benefits that the standard has to offer. So this is where we started to collect the basic information.

### **8.2.1.2 Workshops and courses**

Several of the tool vendors can offer workshops that involve their tools. We encourage the interested ones to apply for these workshops.

***This answers Q3.7 "How does the learning curve look like?"***

Much must be learned in the beginning, but once the toughest parts are conquered, it is manageable to understand the major parts.

## **8.2.2 Introduction of a new standard**

At the start of this thesis, we had no previous experience on the subject of AUTOSAR. Our experience is that the standard is easy to understand as a whole, the relations between the layers of the architecture and the fundamental parts about SWCs. But once a specific part of the standard has to be used or examined, that previous knowledge seems inadequate.

We had problems to understand the whole standard and still have a lack of understanding for some parts. Our experience and situation should be transferable to others with a similar experience level as ours. Our verdict is that the standard is simply too large for a single individual to understand.

The methodology does not define any roles for activities which must be performed to produce certain output [7]. We have experienced and discovered that depending on the users' role, some parts are more important than others. See Table 41 for a summary of the roles along with an explanation for each one and examples of recommended documents.

Roles	Explanation	Document examples
Application developer	This role involves those who have the responsibility to create and develop SWCs. Thus, it is not necessary to understand the whole architecture. The <i>Application Layer</i> is the most important to understand and the role of the <i>RTE Layer</i> is good to know about. How SWCs communicate is very important, especially the difference between <i>Ports</i> and <i>Interfaces</i> . Internal communication within the SWCs' <i>runnables</i> is also good to understand. Basic knowledge of the <i>Methodology</i> is good but not crucial.	[21], [7] and [6]
System creator	Mapping is important for those who have the responsibility to create the system design with the available SWCs. To know about which available nodes there are in the target system in order to <i>extract</i> the correct SWCs. It is not necessary to understand the whole standard, but the <i>Methodology</i> is one of the more important parts. How to connect SWCs with each other and with <i>System Signals</i> requires knowledge about <i>Ports</i> , <i>Interfaces</i> and the role of the <i>RTE</i> .	[19],[19],[22], [17] and [7]
ECU Configuration	One of the larger activities which must be performed by a role is the configuration of the ECU. It is not just important to understand the BSW modules, but also how the actual hardware works. It much easier to configure the modules correctly if the role possesses such knowledge. The interaction between the <i>RTE Layer</i> and <i>BSW Layer</i> is important to understand the flow of data between the layers. The flow of data between BSW modules are even more important, especially <i>PDU</i> s and <i>System Signals</i> .	[6], [74] and specifications of the BSW modules to be used.
Tool developer	For tool developers the most important knowledge is the AUTOSAR <i>Methodology</i> and <i>Templates</i> . Templates describe AUTOSAR objects and how they relate to other AUTOSAR objects. These relations are very important to understand and follow, in order to deal with AUTOSAR objects and files correctly. The <i>Methodology</i> defines activities which are to be performed by tools, so it is important that tool is designed to reproduce the expected behavior.	[7], [3] and depending on the tool to be developed, [BSW Modules' Specifications]
Process responsible	The one who is responsible for the use of AUTOSAR in a project should have good knowledge about the <i>Methodology</i> . What the necessary <i>activities</i> which must be performed are and how these could be integrated into an already existing workflow.	[7], [5] and [3]

Table 41: Roles and interests

*This answers Q1.2 “Is it manageable to understand the whole standard or are some parts addressed for certain roles?” and Q1.3 “What parts of the standard is good to know about depending on the user’s role?”*

The standard is very large and a consequence of this is that a person might have problems to grasp the whole standard. But it is not necessary to fully understand every part of the standard. Depending on the role, some parts are of more interest than others. See Table 41 for examples of recommended documents and parts of the standard depending on the user’s role.

### 8.2.3 Productivity and Cost

There is going to be a high cost initially to adapt to the standard. Employees that are going to work with AUTOSAR projects need the correct experience and knowledge in order to work effectively and create the expected system(s). This could be achieved either by read about the standard and use the available documents from AUTOSAR or preferably send the employees to workshops/lectures which other companies offer. Tools that is ready for AUTOSAR needs to be bought to effectively develop systems. How the productivity and cost figures over time could look like, could be seen in Figure 23, which is a sketched version.

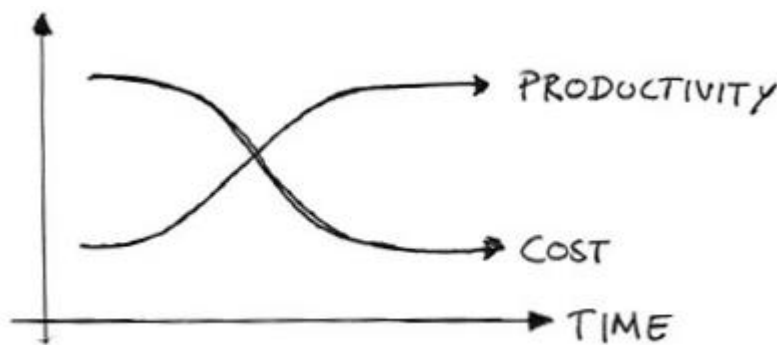


Figure 23: Cost and Productivity over time

To shorten the time with high cost and increase the productivity we recommend the following actions and considerations:

- **Invest time for workshops and lectures** – To learn about AUTOSAR from people that already know much about it is recommended. There are parts of the standard that are more important than other parts. Consulting with expertise is recommended in this case. Developers might be disoriented and unused to newer principals.
- **Select a tool from an established tool vendor** – The tools used to develop AUTOSAR systems are very important. You as a person are not supposed to learn every single step and manually convert all the files from one format to another; this is a work for a tool. The methodology is prepared to utilize tools for certain work packages.
- **Understand the “right” layers** – The AUTOSAR architecture is divided into different layers in order to make upper layers independent of the hardware. If you are a software developer and your target is to develop SWCs, you do not need to know about the whole standard, only the parts that are related to your topic. A person which are going to configure the hardware

and lower BSW modules need to know about the target hardware. This is not a job for the application developers; they can develop without hardware dependency.

- **Make sure the hardware is supported** – At an early stage, investigate which hardware that is supported by the tool, especially tools that are needed when configuring the hardware (BSW Editor). We have been using a tool that have support for our hardware, but is still under development.
- **Do not expect direct benefits** – As stated earlier, an investment must first be made before any benefits can be received. So for the first project or might even projects, no direct benefits should be expected.
- **Increased budget for early projects** – You need the right tools to develop AUTOSAR systems. These tools needs to be bought and that would lead to an increased budget. However, this extra cost should be seen as an investment for upcoming projects, as long as there is going to be future ones.

Before any of the implied benefits AUTOSAR has to offer can be collected, an investment must be made, such as resources and money. As the numbers of projects that involve AUTOSAR increase, so should also the productivity and income.

***This answers Q3.3 “After a migration, how could the expected productivity and cost figures look like?”***

For the first projects the benefits from the standard should be taken at ease, the real repayment from earlier investment should visible in later projects.

## **8.3 Lesson 3: Sharing files between tools**

A couple of the questions this thesis tries to answer are in the subject of file sharing between tools. What was learned regarding this during the development of the demonstrator can be found in this section as lesson 3.

### **8.3.1 What files can you actually share between tools?**

Depending on what you wish to share, there are limitations that need to be considered before you try to share files all over the place.

#### ***8.3.1.1 Summary of the file sharing***

When evaluating the ability to share files between tools three areas were focused on: SWCs, ECU extract and BSW configuration. The impressions from each area are presented below.

##### **8.3.1.1.1 Software Components**

Most of the files could be used by both tools. The problem is when the involved tools does not support equal amount of the standard, in our case, multiple instantiations of SWCs. The special solutions we had to do in some of the implementation are possible to avoid if everything worked as planned. So there is a high degree of reusability and sharing between the tools if only supported parts of the standard is used by both tools, which can be tricky to discover, something we had to experience ourselves.

#### 8.3.1.1.2 ECU Extract

With the combination of our tools, we could not share the ECU Extract between the tools. COQOS-tP has a requirement that all used SWCs must be placed inside a Composition, something that Arctic Studio does not support. Valuable time can be lost if the tools used cannot share the same ECU Extracts, since the same job must be done several times.

#### 8.3.1.1.3 Basic software configuration

Even though it should be theoretical possible to share the configuration of the BSW Modules between tools, it is not done in practice. Vendors have the opportunity to add so called SDGs to the .arxml-files to add special information only used by a specific tool. This makes it problematic to share between different tools, since these tags cannot be interpreted correctly. It is no problem to reuse the configurations within the tools.

### 8.3.2 Not as easy as it might look like

There are mainly two different ways how you can organize your .arxml-files which describes different aspects of the standard. Either you combine all the settings into one big file or separate different modules into simpler files. The tools we have been working with are doing this separation differently from each other; Arctic Studio combines BSW-modules into one single file with all the necessary information and in COQOS-tP the configuration is sliced into smaller and more specific pieces, but also has the possibility to combine everything into larger files.

Both ways has good and bad sides for sure, but it makes it tougher to exchange the information between the two tools.

Some of the files are easier to share; modules that are higher up on the architecture should be easier to share than modules that are hardware specific. Once you move down the architecture, the modules are more and more specific for the target hardware.

#### 8.3.2.1 Arctic Studio

ArcCore have made the choice to combine the basic-software modules in one file that can be edited with their BSW-builder which is a part of the Arctic Studio. There is a possibility to divide the modules into separated files but this is not how the tool is mainly designed for. The validation of the system is one part that is affected negatively by dividing the files. The RTE-module and OS-module are connected with each other and if these modules are placed in different files, the RTE-module will not find some of the necessary settings that can be found in the OS-module.

The solution for this problem could be to divide the modules into different files, but when they are going to be used in a project, they should be imported to a single .arxml file to make the validation more simple and also the generation of the whole system.

The BSW-builder also saves information in the files called ADMIN-DATA which is used by the tool to track for example which version number of the tool that was used when the file was created.

#### 8.3.2.2 COQOS-tP

No limitations regarding splitting of .arxml files were experienced using COQOS-tP. One thing to remember though is that search paths to new .arxml-files have to be added manually to the generator script otherwise they will not be included in the generation step which will result in errors.

## 8.4 Reflections on a good AUTOSAR development tool

During our thesis work we have been using two different types of tools to create our AUTOSAR system. Arctic studio which is based on Eclipse and Artop and QQQOS-tP, a text-based generator tools for Linux. While working with these tools, we have gather thoughts and ideas what we are missing and what we like about the, or could be changed to be even better tools for users. In this section we are going to present our thoughts what we would like to see in a tool.

### 8.4.1 A good development tool

Since we have been working with these tools quite intensive and experienced their pros and cons, we started to summarize and discuss what a “good” tool preferably should consist of. This is what we came up with:

- Based on Artop
  - The main idea behind ARTOP is to make it easier to export and import files. This is fundamental for any tool that handles .arxml-files. There is no need for each tool vendor to implement their own version of this functionality, if they are based on Artop they can compete on other factors instead. Another benefit is that tools based on Artop will be able to handle new versions of the AUTOSAR standard much faster, since Artop is taking care of the releases. For import and export of .arxml-files.[8]
- Graphical interface
  - Designing the system topology together with all the SWCs would easier if you could do this graphically. Creating the ports and interfaces, and then export it as arxml-files. To discuss a solution around a visible object is from our experience something to strive for and would greatly improve the communication between all involved partners. It is easier to recognize flaws in the system etc. This is something we have truly missed.
- Debug
  - Debug simulation and complete tool chain is somewhat interleaved with each other. If it was easy to debug the solution on target (or simulated) many simple programming or logical errors could be found and corrected fairly easy. If you have downloaded the software to the target and are not able to see what is actually going on, you are in some deep water. When we started to download small solutions to the hardware we had no tool to debug the hardware, which we cannot recommend to anyone.
- Simulation
  - Simulation of single SWCs with predefined input signals together with expected output signals and actual signals is something we are missing. This would make it much easier to run tests and see that the SWCs behave as designed. None of the tools we have been working with has the possibility to simulate the system today. ArcCore has plans to offer simulation possibilities in the future, but nothing more is decided. There are other companies on the market with tools which supports simulation of the VFB, for example Mecel [28]and Vector [38]. Our system was small enough to be handled without simulation, but for larger systems, simulation is preferred.
- Mature



- We would recommend using a tool that is established. We have discovered bugs while using the tools, especially in Arctic Studio since we have been working with their bleeding edge version of the tool, and that was something we had in mind while using the tool and therefore we are forgiving when we found errors. There were times we could not figure out if we made the mistakes or if the tool did not work as intended. However, the market is still young but there are likely to emerge tools from the larger companies which could be seen as mature.
- HW support
  - It should be noted that you are free to use any IDE to create the behavior for the SWCs. The tool needs to support the target hardware in order to configure it properly. Investigate early what kind of hardware that is supported, vendors offer different hardware solutions and mean to configure these. A mature tool with full support for the target hardware is recommended. We encountered this problem at the start of the thesis when Arctic Studio did not have support for our development board. Therefore we had to work with bleeding edge versions of the core and the tools, which sometimes resulted in problems caused by bugs in the tools.
- Documentation
  - With maturity comes documentation. Online manuals or some other way to fully understand the tool is recommended. You might understand quite a lot of the AUTOSAR standard, but when it comes to certain details, there is a high chance to get lost. How is the tool intended to be used or what type of settings can you set in this part of the configuration. We have really missed proper documentation when using our selected tools. Arctic Studio has documentation but we have not been able to access these since we are not a paying customer.
- Workflow
  - If the tools consists of several tools in a suite a clear workflow between these. What are the necessary steps I must take in order to create my systems? Arctic Studio has almost nailed it. They present a workflow for their sub tools that are included in the Arctic Studio, in what order things are supposed to be done. However it is not perfect since you have to go back and forth in a non linear way sometimes depending on the system you are about to setup.
- C standard
  - In automotive industry or other similar fields, C89 has almost become a de facto standard. It is for example widely used in safety-critical systems. If the tool are designed to generate C89 code it is preferred.
- Examples
  - More examples of small systems together with documentation would increase the understanding of the tool and also make it easier to use it. Small examples that isolate smaller parts and functionality that is deserved. For example I/O, Com, Sensor/Actuator.
- Complete chain
  - There are some vendors that offers tools that has a single function in the AUTOSAR chain and others that offers a complete suite that handles all the steps in the methodology. From design to deployment. It is our belief that a complete suite is

better to use since you do not need to learn several tools, the workflow can be optimized and files should be easier to share between them.

- File comparison
  - Most tools to check for differences between files are based on text differences. That is usually great when you are working with files that only contain code or plain text. One thing we have been missing is a good tool to compare .arxml-files and only highlight the vital differences between the files. For example if you have two files with the same SWC, but the structure is changed in one of the files due to a different workflow, it is almost impossible to see the relevant changes. So instead of comparing the files on text level, we propose a solution where blocks of text are compared instead and the possibility to select which parts of the files you wish to compare. One case is that you know that the name of the components is going to be different, but you are interested to know the differences for a certain property. We have missed this functionality during the thesis.

## 8.5 Reflection on Configuration and Implementation

One thing that stood out among the most once the demonstrator was completed was how little time was actually spent implementing the functionality. If the CAN modules that had to be implemented for COQOS-tP and the customized receiving of button pushes from the STM32 board are not considered the only C-code necessary to write was for the implementations of the SWCs. If the total time spent on implementing the SWCs was to be summarized it would not be more than an afternoon. One interesting fact is that the code to facilitate debugging outputs in the terminal on Linux consisted of more lines of code than the actual functionality of the components. See Appendix 2: Demonstrator software components implementations.

Configuring the whole demonstrator on the other hand was quite time consuming even though it was a relatively painless process. The reason why the configuration was such a major task is because there are so many parts (SWCs, RTE, BSW etc.) required in setting up a complete system including the communication stack.

One thing to remember though is that the complexity of the SWCs obviously plays a part in how much time needs to be allocated for implementation; nevertheless based on the demonstrator created in this thesis it is easy to get the impression that developers will spend more time configuring their systems than implementing the functionality.

***This answers Q3.9 "Is the development paradigm shifted from implementation towards configuration?" and Q3.10 "After a migration, is most time spent on Configuration or Implementation?"***

Most time is indeed spent on configuration rather than implementation. In our case we had to implement also, but if the tools could generate all the necessary files, the extra implementation could be converted into configuration instead.

## 8.6 Reflection on simulation possibilities

The AUTOSAR architecture (section 2.4), Virtual Functional Bus (mentioned briefly in section 2.5) and SWCs (section 2.5) offers good possibilities for simulation.

### 8.6.1 Communication between Software Components

The architecture consists of multiple layers and each layer should be possible to simulate. However, instead of simulating all layers, it would probably be more beneficial and easier to simulate the Virtual Functional Bus (realized as the RTE and BSW layers). Only the communication between SWCs are represented by using the Virtual Functional Bus, not how the components are actually mapped. Since the SWCs are prepared to be hardware independent, this communication should be fairly easy to simulate.

### 8.6.2 Behavior of Software Components

The SWCs have *Ports* with predefined *Interfaces* and are modeled either as atomic components or as compositions. They are therefore excellent candidates to be simulated and tested with a workbench. A creation of a workbench should be straightforward. The workbench should then connect to the *RPorts* and *PPorts* on the component to be tested in order to so send input data to the component, and read the output data, see Figure 24. This would be a great way to test the components behavior responds correctly for a given set of input values.

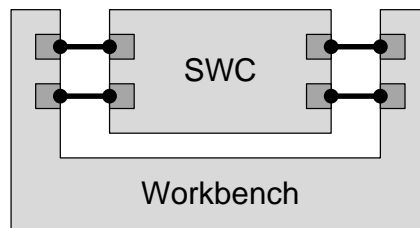


Figure 24: Sketch of a workbench to test SWCs

#### ***This answers Q1.7 “How does the standard support simulation?”***

Thanks to the introduction of a layered architecture and the VFB. There are good simulation possibilities for the communication between SWCs. There are also good chances to test the behavior of the components thanks to the hardware independence and ports.

## 8.7 Reflection on suitable CrossControl products to be used with AUTOSAR

COQOS-tP could be a candidate to be used on CrossControl’s display computers as long as the system is developed on the same platform as the target. In addition, since COQOS-tP does not currently have generator support for the necessary BSW modules in order to access CAN all CAN modules would have to be implemented manually which would eat up a lot time and resources. The implementation done for the demonstrator in this thesis (section 6) is only a small part of what the AUTOSAR standard defines and still a lot of time was spent on that task.

Arc Core do not at this point provide hardware support for the hardware used by CrossControl’s products; both with the microcontroller architecture and memory requirements in mind. If this limitation did not exist it is believed an AUTOSAR implementation created in Arctic Studio would be

the preferred choice over COQOS-tP on a non-display ECU since there is less work involved in getting the CAN communication operational.

***This answers Q3.2 “Are any of CrossControl’s products suitable to be used with the AUTOSAR standard?”***

Neither tool suite evaluated in this thesis is at this point recommended or possible to use when developing an AUTOSAR solution for CrossControl’s products. Since COQOS-tP lacks generation support for the bottom part of the communication stack it is only an option if the missing part is implemented manually which would be very time-consuming. Arctic Studio lacks the necessary hardware support in order to be placed on any of CrossControl’s products.

## 9 Conclusions

In this thesis, questions regarding the AUTOSAR standard, tools used to develop AUTOSAR compliant solutions and migration from existing solutions has been answered. A demonstrator which was used to answers some of these questions was developed according to the rules set by the AUTOSAR standard. Two development tools have been used during the thesis, namely Arctic Studio from ArcCore and COQOS-tP from OpenSynergy.

Three main topics have been the centre for our question formulations and in the next three following sections, answers to these topics will be presented.

### 9.1 The standard

The standard has many benefits to offer for those willing to grasp the standard. The goals with hardware independence for the applications are truly possible. Migration from existing solutions has been taken into account when the standard was created, especially due to the introduction of *Complex Drivers*. The standard itself is large and it growing steadily, this could be a problem since it is tough to understand and get a complete view of the whole standard. The flow of data is a corner stone which should be understood.

### 9.2 The tools

It was discovered that it is very important that the development tools used to configure the BSW have support for the target hardware. Our experience is that a workflow involving development tools from different tool vendors are not encouraged if the involved tools does not support the same amount of the standard. Even though the format of the description files is standardized, it is problematic to share files between tools from different tool vendors.

### 9.3 Migration

AUTOSAR pushes the paradigm from implementations towards configuration; more time is spent on configuration rather than implementation. The possible benefits which are achievable with AUTOSAR standard should not be expected for the first projects involving the standard since there is going to be a steep learning curve at the beginning. After a migration, there is a risk that the previously used hardware is not powerful enough to run the migrated solution due to an increased use of memory and execution overhead or the hardware is not supported by the development tools.

### 9.4 Future work

One approach of modeling the AUTOSAR application layer not explored in this thesis is doing it graphically. Several tools on the market provide this feature and it would be interesting to evaluate what advantages and disadvantages may exist in terms of for example efficiency.

Conclusions regarding the support for migration of non-AUTOSAR systems to AUTOSAR were based on what the AUTOSAR standard declare in theory. But since the impression is that other parts of the standard are not as simple in practice as they seem in theory it would be of interest to migrate a real system to AUTOSAR and find out if theoretically based impressions hold true in practice as well.

## 10 References

- [1] Christof Ebert and Capers Jones, "Embedded Software: Facts, Figures, and Future," 2009.
- [2] AUTOSAR. (2011, February) FAQ. [Online]. <http://www.autosar.org>
- [3] AUTOSAR GbR, "Technical Overview," 2008.
- [4] AUTOSAR. (2010, September) TECHNICAL OVERVIEW. [Online]. <http://www.autosar.org/>
- [5] AUTOSAR, "AUTOSAR\_Tutorial," 2007.
- [6] AUTOSAR, "Layered Software Architecture," 2009.
- [7] AUTOSAR, "AUTOSAR Methodology," 2008.
- [8] Simon Fürst, "AUTOSAR – A Worldwide Standard is on the Road.,".
- [9] Autosar, "Explanation of Application Inter-faces of the Body and Comfort Domain," 2009. [Online]. <http://www.autosar.org>
- [10] AUTOSAR. (2011, January) Application Interfaces. [Online]. <http://www.autosar.org/index.php?p=3&up=1&uup=6&uuup=1&uuuup=0&uuuuup=0>
- [11] AUTOSAR, "Explanation of Application Interfaces of Occupant and Pedestrian Safety Systems Do-main," 2009.
- [12] AUTOSAR, "Explanation of Application Interfaces of the Body and Comfort Domain," 2009.
- [13] AUTOSAR, "Explanation of Application Interfaces of the Chassis Domain," 2009.
- [14] AUTOSAR, "Explanation of Application Interfaces of the HMI, Multimedia and Telematics Domain," 2009.
- [15] AUTOSAR, "Explanation of Application Interfaces of the Powertrain Domain," 2009.
- [16] Renesas. (2010, October) Why does the automotive industry need AUTOSAR? [Online]. <http://www2.renesas.eu/applications/automotive/ee/autosar/index.html>
- [17] AUTOSAR, "Specification of RTE," 2009.
- [18] D Schreiner and K.M. Goschka. (2007) A Component Model for the AUTOSAR Virtual Function Bus.
- [19] AUTOSAR GbR, "Specification of the Virtual Funtional Bus," 2008.
- [20] Sommerville, *Software Engineering.*: Addison Wesley, 2006.

- [21] AUTOSAR, "Software Component Template," 2009.
- [22] AUTOSAR GbR, "UML Profile for AUTOSAR," 2006.
- [23] Alberto Ferrari, Marco Di Natale, Giacomo Gentile, Giovanni Reggiani, and Paolo Gai, "Time and memory tradeoffs in the implementation of AUTOSAR components," 2009.
- [24] EE Times. (2011, March) AUTOSAR architecture expands safety and security applications. [Online]. <http://www.eetimes.com/design/automotive-design/4213069/AUTOSAR-architecture-expands-safety-and-security-applications>
- [25] Wikipedia. (2011, February) ISO 26262. [Online]. [http://en.wikipedia.org/w/index.php?title=ISO\\_26262&oldid=415873991](http://en.wikipedia.org/w/index.php?title=ISO_26262&oldid=415873991)
- [26] Dirk Diekhoff, "AUTOSAR Tooling in practice," 2010.
- [27] ElektroBit. (2010, October) EB Tresos. [Online]. <http://www.eb-tresos-blog.com/solutions/tresos/>
- [28] Mecel. (2010, October) Mecel Picea. [Online]. <http://www.mecel.se/products/mecel-picea>
- [29] Mentor Graphics. AUTOSAR Products. [Online]. <http://www.mentor.com/products/vnd/autosar-products/>
- [30] Infineon. (2010, October) Infineon AUTOSAR Software. [Online]. <http://www.infineon.com/cms/en/product/channel.html?channel=db3a30431b3e89eb011b4aac01f07b7d>
- [31] Continental. (2010, October) Configuration Tooling. [Online]. [http://www.conti-online.com/generator/www/de/en/continental/engineering\\_services/general/autosar/autosar\\_tooling\\_en.html](http://www.conti-online.com/generator/www/de/en/continental/engineering_services/general/autosar/autosar_tooling_en.html)
- [32] BOSCH India. (2010, October) AUTOSAR Products and Services. [Online]. <http://www.boschindia.com/content/language1/html/14478.htm>
- [33] Wikipedia. (2011, January) Robert Bosch GmbH. [Online]. [http://en.wikipedia.org/w/index.php?title=Robert\\_Bosch\\_GmbH&oldid=421405469](http://en.wikipedia.org/w/index.php?title=Robert_Bosch_GmbH&oldid=421405469)
- [34] pulse-AR. (2010, October) Quasar. [Online]. <http://www.pulse-ar.com/en/products>
- [35] dSpace. (2010, October) SystemDesk. [Online]. [http://www.dspaceinc.com/en/inc/home/products/sw/system\\_architecture\\_software/systemdesk.cfm](http://www.dspaceinc.com/en/inc/home/products/sw/system_architecture_software/systemdesk.cfm)
- [36] dSpace. (2010, October) TargetLink. [Online]. <http://www.dspaceinc.com/en/inc/home/products/sw/pcgs/targetli.cfm>

- [37] Vector. (2010, October) Microsar. [Online]. [http://www.vector.com/vi\\_microsar\\_en.html](http://www.vector.com/vi_microsar_en.html)
- [38] Vector. (2010, October) AUTOSAR Tools. [Online].  
[https://www.vector.com/vi\\_autosar\\_tools\\_en,223.html](https://www.vector.com/vi_autosar_tools_en,223.html)
- [39] Geensoft. (2010, October) AUTOSAR Builder. [Online].  
<http://www.geensoft.com/en/article/autosarbuilder>
- [40] ETAS. (2010, October) Etas. [Online].  
[http://www.etas.com/en/products/applications\\_standards\\_autosar\\_ascet\\_md\\_modeling\\_design.php](http://www.etas.com/en/products/applications_standards_autosar_ascet_md_modeling_design.php)
- [41] ETAS. (2010, October) ETAS. [Online].  
[http://www.etas.com/en/products/applications\\_standards\\_autosar\\_ascet\\_rp\\_rapid\\_prototyping.php](http://www.etas.com/en/products/applications_standards_autosar_ascet_rp_rapid_prototyping.php)
- [42] ETAS. (2010, October) ETAS. [Online].  
[http://www.etas.com/en/products/applications\\_standards\\_autosar\\_ascet\\_se\\_software\\_engineering.php](http://www.etas.com/en/products/applications_standards_autosar_ascet_se_software_engineering.php)
- [43] ETAS. (2010, October) ETAS. [Online].  
[http://www.etas.com/en/products/applications\\_standards\\_autosar\\_ascet\\_scm.php](http://www.etas.com/en/products/applications_standards_autosar_ascet_scm.php)
- [44] ETAS. (2010, October) ETAS. [Online].  
[http://www.etas.com/en/products/applications\\_standards\\_autosar\\_ascet\\_diff.php](http://www.etas.com/en/products/applications_standards_autosar_ascet_diff.php)
- [45] KPIT Cummins. (2010, October) KPIT Cummins. [Online].  
[http://www.kpitcummins.com/ats/automotiveelectronics/offerings\\_autosar.htm](http://www.kpitcummins.com/ats/automotiveelectronics/offerings_autosar.htm)
- [46] MathWorks. (2010, October) MathWorks. [Online].  
<http://www.mathworks.com/automotive/standards/autosar.html>
- [47] ArcCore. (2010, October) BSW builder. [Online]. <http://download.arccore.com/bswBuilder.pdf>
- [48] ArcCore. (2010, October) RTE Builder. [Online]. <http://download.arccore.com/rteBuilder.pdf>
- [49] ArcCore. (2010, October) SWC Builder. [Online]. <http://download.arccore.com/swcBuilder.pdf>
- [50] ArcCore. (2010, October) Extract Builder. [Online]. <http://download.arccore.com/extBuilder.pdf>
- [51] ArcCore. (2010, October) ArcCore. [Online]. <http://arccore.com/products>
- [52] OpenSynergy, "COQOS-tP\_Datasheet\_en.pdf," 2010. [Online].  
[http://www.opensynergy.com/sites/default/files/media/sonstige/COQOS-tP\\_Datasheet\\_en.pdf](http://www.opensynergy.com/sites/default/files/media/sonstige/COQOS-tP_Datasheet_en.pdf)
- [53] OpenSynergy, "COQOS-tC\_Datasheet\_en.pdf," October, 2010. [Online].  
[http://www.opensynergy.com/sites/default/files/media/sonstige/COQOS-tC\\_Datasheet\\_en.pdf](http://www.opensynergy.com/sites/default/files/media/sonstige/COQOS-tC_Datasheet_en.pdf)



- [54] Michael Rudorfer, Stefan Voget, and Stephan Eberle, "Artop Whitepaper," 2010.
- [55] dSpace. (2010, October) Cooperation with EB Automotive. [Online].  
[http://www.dspaceinc.com/en/inc/home/products/our\\_solutions\\_for/autosaratdspace/autosar\\_coop.cfm](http://www.dspaceinc.com/en/inc/home/products/our_solutions_for/autosaratdspace/autosar_coop.cfm)
- [56] ETAS. (2010, October) ETAS. [Online].  
[http://www.etas.com/en/products/applications\\_iso26262.php](http://www.etas.com/en/products/applications_iso26262.php)
- [57] AUTOSAR, "List of Basic Software Modules," 2009.
- [58] OpenSynergy, "COQOS RTE Generator User Manual (v2.3.0)," 2010.
- [59] STMicroelectronics. (2010, November) STM3210E-EVAL. [Online].  
<http://www.st.com/internet/evalboard/product/204176.jsp>
- [60] (2010, December) Qt. [Online]. <http://qt.nokia.com/>
- [61] (2010, December) wxWidgets. [Online]. <http://www.wxwidgets.org/>
- [62] (2010, December) Qwt. [Online]. <http://qwt.sourceforge.net/>
- [63] AUTOSAR, "Specification of CAN Driver," 2010.
- [64] AUTOSAR, "Specification of CAN Interface," 2010.
- [65] AUTOSAR, "Specification of BSW Scheduler," 2008.
- [66] Atollic. (2010, November) TrueSTUDIO. [Online]. <http://www.atollic.com/index.php/truestudio>
- [67] AUTOSAR GbR, "Specification of Operating System," 2008.
- [68] ArcCore. (2010, November) Forum - Porting Hc(s)o8. [Online].  
<http://arccore.com/forum/viewtopic.php?f=2&t=66>
- [69] IAR Systems. (2010, November) ISO/ANSI C compliance. [Online].  
<http://www.iar.com/website1/1.0.1.0/466/1/>
- [70] Vector. (2010, November) ECU Software Catalog. [Online].  
[http://www.vector.com/portal/medien/e\\_media/en/ecusoftware\\_catalog/WebSearch/page0058.html](http://www.vector.com/portal/medien/e_media/en/ecusoftware_catalog/WebSearch/page0058.html)
- [71] ElektroBit. (2010, Oktober) [Online]. [http://www.eb-tresos-blog.com/wp-content/uploads/2010/06/Datasheet\\_EBtresos\\_autocore\\_en\\_screen.pdf](http://www.eb-tresos-blog.com/wp-content/uploads/2010/06/Datasheet_EBtresos_autocore_en_screen.pdf)
- [72] AUTOSAR, "Generic Structure Template," 2009.

[73] Daehyun Kum, Gwang-Min Park, Seonghun Lee, and Wooyoung Jung, "AUTOSAR Migration from Existing Automotive Software," 2008.

[74] AUTOSAR, "Specification of Communication," 2010.

[75] AUTOSAR, "Specification of Communication Stack Types," 2010.

## 11 Appendix 1: Demonstrator ARXML models

### 11.1 HeatRegulator.arxml

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/3.1.4"><TOP-LEVEL-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>CrossControl</SHORT-NAME>
    <SUB-PACKAGES>
      <AR-PACKAGE>
        <SHORT-NAME>SoftwareComponents</SHORT-NAME>
        <ELEMENTS>
          <APPLICATION-SOFTWARE-COMPONENT-TYPE>
            <SHORT-NAME>HeatRegulator</SHORT-NAME>
            <PORTS>
              <R-PORT-PROTOTYPE>
                <SHORT-NAME>Position</SHORT-NAME>
                <REQUIRED-COM-SPECS>
                  <CLIENT-COM-SPEC>
                    <OPERATION-REF DEST="OPERATION-
PROTOTYPE"/>CrossControl/Interfaces/IRegulatorPosition/SetPosition</OPERATION-REF>
                  </CLIENT-COM-SPEC>
                  </REQUIRED-COM-SPECS>
                  <REQUIRED-INTERFACE-TREF DEST="CLIENT-SERVER-
INTERFACE"/>CrossControl/Interfaces/IRegulatorPosition</REQUIRED-INTERFACE-TREF>
                </R-PORT-PROTOTYPE>
                <R-PORT-PROTOTYPE>
                  <SHORT-NAME>RegulatorIO</SHORT-NAME>
                  <REQUIRED-COM-SPECS>
                    <UNQUEUED-RECEIVER-COM-SPEC>
                      <DATA-ELEMENT-REF DEST="DATA-ELEMENT-
PROTOTYPE"/>CrossControl/Interfaces/IRegulatorIO/RegulatorValue</DATA-ELEMENT-REF>
                      <INIT-VALUE-REF DEST="INTEGER-
LITERAL"/>CrossControl/Interfaces/ConstInitInt32/InitInt32</INIT-VALUE-REF>
                    </UNQUEUED-RECEIVER-COM-SPEC>
                    </REQUIRED-COM-SPECS>
                    <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-
INTERFACE"/>CrossControl/Interfaces/IRegulatorIO</REQUIRED-INTERFACE-TREF>
                  </R-PORT-PROTOTYPE>
                </PORTS>
              </APPLICATION-SOFTWARE-COMPONENT-TYPE>
              <SWC-IMPLEMENTATION>
                <SHORT-NAME>HeatRegulatorImpl</SHORT-NAME>
                <BEHAVIOR-REF DEST="INTERNAL-
BEHAVIOR"/>CrossControl/SoftwareComponents/HeatRegulatorInternalBhv</BEHAVIOR-REF>
              </SWC-IMPLEMENTATION>
              <INTERNAL-BEHAVIOR>
                <SHORT-NAME>HeatRegulatorInternalBhv</SHORT-NAME>
                <COMPONENT-REF DEST="APPLICATION-SOFTWARE-COMPONENT-
TYPE"/>CrossControl/SoftwareComponents/HeatRegulator</COMPONENT-REF>
                <EVENTS>
                  <TIMING-EVENT>
                    <SHORT-NAME>HeatRegulatorMainTrigger</SHORT-NAME>
                    <START-ON-EVENT-REF DEST="RUNNABLE-
ENTITY"/>CrossControl/SoftwareComponents/HeatRegulatorInternalBhv/HeatRegulatorMain</START-ON-
EVENT-REF>
                    <PERIOD>1.0</PERIOD>
                  </TIMING-EVENT>
                </EVENTS>
                <RUNNABLES>
                  <RUNNABLE-ENTITY>
                    <SHORT-NAME>HeatRegulatorMain</SHORT-NAME>
                    <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
                    <DATA-READ-ACCESS>
                      <DATA-READ-ACCESS>
                        <SHORT-NAME>RegulatorIOReadAccess</SHORT-NAME>
                        <DATA-ELEMENT-IREF>
                          <R-PORT-PROTOTYPE-REF DEST="R-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/HeatRegulator/RegulatorIO</R-PORT-PROTOTYPE-REF>
                        <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-
PROTOTYPE"/>CrossControl/Interfaces/IRegulatorIO/RegulatorValue</DATA-ELEMENT-PROTOTYPE-REF>
                      </DATA-ELEMENT-IREF>
                    </DATA-READ-ACCESS>
                  </DATA-READ-ACCESS>
                </RUNNABLE-ENTITY>
              </RUNNABLES>
            </INTERNAL-BEHAVIOR>
          </APPLICATION-SOFTWARE-COMPONENT-TYPE>
        </ELEMENTS>
      </AR-PACKAGE>
    </SUB-PACKAGES>
  </AR-PACKAGE>
</TOP-LEVEL-PACKAGES>
</AUTOSAR>
```

```

        <SERVER-CALL-POINTS>
        <SYNCHRONOUS-SERVER-CALL-POINT>
        <SHORT-NAME>PositionCallPoint</SHORT-NAME>
        <OPERATION-IREFS>
        <OPERATION-IREF>
        <R-PORT-PROTOTYPE-REF DEST="R-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/HeatRegulator/Position</R-PORT-PROTOTYPE-REF>
        <OPERATION-PROTOTYPE-REF DEST="OPERATION-
PROTOTYPE"/>CrossControl/Interfaces/IRegulatorPosition/SetPosition</OPERATION-PROTOTYPE-REF>
        </OPERATION-IREF>
        </OPERATION-IREFS>
        </SYNCHRONOUS-SERVER-CALL-POINT>
        </SERVER-CALL-POINTS>
        <SYMBOL>HeatRegulatorMain</SYMBOL>
        </RUNNABLE-ENTITY>
        </RUNNABLES>
        <SUPPORTS-MULTIPLE-INSTITIATION>>false</SUPPORTS-MULTIPLE-INSTITIATION>
        </INTERNAL-BEHAVIOR>
        </ELEMENTS>
        </AR-PACKAGE>
        </SUB-PACKAGES>
        </AR-PACKAGE>
    </TOP-LEVEL-PACKAGES></AUTOSAR>

```

## 11.2 SeatHeater.arxml

```

<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/3.1.4"><TOP-LEVEL-PACKAGES>
    <AR-PACKAGE>
        <SHORT-NAME>CrossControl</SHORT-NAME>
        <SUB-PACKAGES>
            <AR-PACKAGE>
                <SHORT-NAME>SoftwareComponents</SHORT-NAME>
                <ELEMENTS>
                    <APPLICATION-SOFTWARE-COMPONENT-TYPE>
                        <SHORT-NAME>SeatHeater</SHORT-NAME>
                        <PORTS>
                            <P-PORT-PROTOTYPE>
                                <SHORT-NAME>Levels</SHORT-NAME>
                                <PROVIDED-COM-SPECS>
                                    <SERVER-COM-SPEC>
                                        <OPERATION-REF DEST="OPERATION-
PROTOTYPE"/>CrossControl/Interfaces/IHeaterLevel/SetHeat</OPERATION-REF>
                                        <QUEUE-LENGTH>1</QUEUE-LENGTH>
                                    </SERVER-COM-SPEC>
                                </PROVIDED-COM-SPECS>
                                <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-
INTERFACE"/>CrossControl/Interfaces/IHeaterLevel</PROVIDED-INTERFACE-TREF>
                            </P-PORT-PROTOTYPE>
                            <P-PORT-PROTOTYPE S="">
                                <SHORT-NAME>LeftSeatHeaterIO</SHORT-NAME>
                                <PROVIDED-COM-SPECS>
                                    <UNQUEUED-SENDER-COM-SPEC>
                                        <DATA-ELEMENT-REF DEST="DATA-ELEMENT-
PROTOTYPE"/>CrossControl/Interfaces/IHeaterIO/HeaterValue</DATA-ELEMENT-REF>
                                        <INIT-VALUE-REF DEST="INTEGER-
LITERAL"/>CrossControl/Interfaces/ConstInitInt32/InitInt32</INIT-VALUE-REF>
                                    </UNQUEUED-SENDER-COM-SPEC>
                                </PROVIDED-COM-SPECS>
                                <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-
INTERFACE"/>CrossControl/Interfaces/IHeaterIO</PROVIDED-INTERFACE-TREF>
                            </P-PORT-PROTOTYPE>
                            <P-PORT-PROTOTYPE>
                                <SHORT-NAME>RightSeatHeaterIO</SHORT-NAME>
                                <PROVIDED-COM-SPECS>
                                    <UNQUEUED-SENDER-COM-SPEC>
                                        <DATA-ELEMENT-REF DEST="DATA-ELEMENT-
PROTOTYPE"/>CrossControl/Interfaces/IHeaterIO/HeaterValue</DATA-ELEMENT-REF>
                                        <INIT-VALUE-REF DEST="INTEGER-
LITERAL"/>CrossControl/Interfaces/ConstInitInt32/InitInt32</INIT-VALUE-REF>
                                    </UNQUEUED-SENDER-COM-SPEC>
                                </PROVIDED-COM-SPECS>
                                <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-
INTERFACE"/>CrossControl/Interfaces/IHeaterIO</PROVIDED-INTERFACE-TREF>
                            </P-PORT-PROTOTYPE>
                        </PORTS>

```

```

        </APPLICATION-SOFTWARE-COMPONENT-TYPE>
        <SWC-IMPLEMENTATION>
            <SHORT-NAME>SeatHeaterImpl</SHORT-NAME>
            <BEHAVIOR-REF DEST="INTERNAL-
BEHAVIOR"/>CrossControl/SoftwareComponents/SeatHeaterInternalBhv</BEHAVIOR-REF>
        </SWC-IMPLEMENTATION>
        <INTERNAL-BEHAVIOR>
            <SHORT-NAME>SeatHeaterInternalBhv</SHORT-NAME>
            <COMPONENT-REF DEST="APPLICATION-SOFTWARE-COMPONENT-
TYPE"/>CrossControl/SoftwareComponents/SeatHeater</COMPONENT-REF>
            <EVENTS>
                <OPERATION-INVOKED-EVENT UUID="">
                    <SHORT-NAME>SetHeatInvokedEvent</SHORT-NAME>
                    <START-ON-EVENT-REF DEST="RUNNABLE-
ENTITY"/>CrossControl/SoftwareComponents/SeatHeaterInternalBhv/SeatHeaterMain</START-ON-EVENT-
REF>
                    <OPERATION-IREF>
                        <P-PORT-PROTOTYPE-REF DEST="P-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeater/Levels</P-PORT-PROTOTYPE-REF>
                        <OPERATION-PROTOTYPE-REF DEST="OPERATION-
PROTOTYPE"/>CrossControl/Interfaces/IHeaterLevel/SetHeat</OPERATION-PROTOTYPE-REF>
                    </OPERATION-IREF>
                </OPERATION-INVOKED-EVENT>
            </EVENTS>
            <RUNNABLES>
                <RUNNABLE-ENTITY>
                    <SHORT-NAME>SeatHeaterMain</SHORT-NAME>
                    <CAN-BE-INVOKED-CONCURRENTLY>true</CAN-BE-INVOKED-CONCURRENTLY>
                    <DATA-WRITE-ACCESS>
                        <DATA-WRITE-ACCESS>
                            <SHORT-NAME>LeftSeatHeaterIOWriteAccess</SHORT-NAME>
                            <DATA-ELEMENT-IREF>
                                <P-PORT-PROTOTYPE-REF DEST="P-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeater/LeftSeatHeaterIO</P-PORT-PROTOTYPE-REF>
                                <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-
PROTOTYPE"/>CrossControl/Interfaces/IHeaterIO/HeaterValue</DATA-ELEMENT-PROTOTYPE-REF>
                            </DATA-ELEMENT-IREF>
                        </DATA-WRITE-ACCESS>
                        <DATA-WRITE-ACCESS>
                            <SHORT-NAME>RightSeatHeaterIOWriteAccess</SHORT-NAME>
                            <DATA-ELEMENT-IREF>
                                <P-PORT-PROTOTYPE-REF DEST="P-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeater/RightSeatHeaterIO</P-PORT-PROTOTYPE-
REF>
                                <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-
PROTOTYPE"/>CrossControl/Interfaces/IHeaterIO/HeaterValue</DATA-ELEMENT-PROTOTYPE-REF>
                            </DATA-ELEMENT-IREF>
                        </DATA-WRITE-ACCESS>
                    </DATA-WRITE-ACCESS>
                    <SYMBOL>SeatHeaterMain</SYMBOL>
                </RUNNABLE-ENTITY>
            </RUNNABLES>
            <SUPPORTS-MULTIPLE-INSTITANTIATION>>false</SUPPORTS-MULTIPLE-INSTITANTIATION>
        </INTERNAL-BEHAVIOR>
    </ELEMENTS>
</AR-PACKAGE>
</SUB-PACKAGES>
</AR-PACKAGE>
</TOP-LEVEL-PACKAGES></AUTOSAR>

```

## 11.3 SeatHeatingController.arxml

```

<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/3.1.4"><TOP-LEVEL-PACKAGES>
    <AR-PACKAGE>
        <SHORT-NAME>CrossControl</SHORT-NAME>
        <SUB-PACKAGES>
            <AR-PACKAGE>
                <SHORT-NAME>SoftwareComponents</SHORT-NAME>
                <ELEMENTS>
                    <APPLICATION-SOFTWARE-COMPONENT-TYPE>
                        <SHORT-NAME>SeatHeatingController</SHORT-NAME>
                        <PORTS>
                            <P-PORT-PROTOTYPE>
                                <SHORT-NAME>RegulatorPosition</SHORT-NAME>
                                <PROVIDED-COM-SPECS>

```

```

        <SERVER-COM-SPEC>
        <OPERATION-REF DEST="OPERATION-
PROTOTYPE"/>CrossControl/Interfaces/IRegulatorPosition/SetPosition</OPERATION-REF>
        <QUEUE-LENGTH>1</QUEUE-LENGTH>
        </SERVER-COM-SPEC>
    </PROVIDED-COM-SPECS>
    <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-
INTERFACE"/>CrossControl/Interfaces/IRegulatorPosition</PROVIDED-INTERFACE-TREF>
    </P-PORT-PROTOTYPE>
    <R-PORT-PROTOTYPE S="">
        <SHORT-NAME>HeaterLevels</SHORT-NAME>
        <REQUIRED-COM-SPECS>
        <CLIENT-COM-SPEC>
        <OPERATION-REF DEST="OPERATION-
PROTOTYPE"/>CrossControl/Interfaces/IHeaterLevel/SetHeat</OPERATION-REF>
        </CLIENT-COM-SPEC>
        </REQUIRED-COM-SPECS>
        <REQUIRED-INTERFACE-TREF DEST="CLIENT-SERVER-
INTERFACE"/>CrossControl/Interfaces/IHeaterLevel</REQUIRED-INTERFACE-TREF>
        </R-PORT-PROTOTYPE>
    </R-PORT-PROTOTYPE>
        <SHORT-NAME>LeftSeatStatus</SHORT-NAME>
        <REQUIRED-COM-SPECS>
        <UNQUEUED-RECEIVER-COM-SPEC>
        <DATA-ELEMENT-REF DEST="DATA-ELEMENT-
PROTOTYPE"/>CrossControl/Interfaces/ISeatStatus/PassengerOnSeat</DATA-ELEMENT-REF>
        <INIT-VALUE-REF DEST="BOOLEAN-
LITERAL"/>CrossControl/Interfaces/ConstInitBoolean/InitBoolean</INIT-VALUE-REF>
        </UNQUEUED-RECEIVER-COM-SPEC>
        </REQUIRED-COM-SPECS>
        <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-
INTERFACE"/>CrossControl/Interfaces/ISeatStatus</REQUIRED-INTERFACE-TREF>
        </R-PORT-PROTOTYPE>
    </R-PORT-PROTOTYPE S="">
        <SHORT-NAME>RightSeatStatus</SHORT-NAME>
        <REQUIRED-COM-SPECS>
        <UNQUEUED-RECEIVER-COM-SPEC>
        <DATA-ELEMENT-REF DEST="DATA-ELEMENT-
PROTOTYPE"/>CrossControl/Interfaces/ISeatStatus/PassengerOnSeat</DATA-ELEMENT-REF>
        <INIT-VALUE-REF DEST="BOOLEAN-
LITERAL"/>CrossControl/Interfaces/ConstInitBoolean/InitBoolean</INIT-VALUE-REF>
        </UNQUEUED-RECEIVER-COM-SPEC>
        </REQUIRED-COM-SPECS>
        <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-
INTERFACE"/>CrossControl/Interfaces/ISeatStatus</REQUIRED-INTERFACE-TREF>
        </R-PORT-PROTOTYPE>
    </PORTS>
</APPLICATION-SOFTWARE-COMPONENT-TYPE>
<INTERNAL-BEHAVIOR>
    <SHORT-NAME>SeatHeatingControllerInternalBhv</SHORT-NAME>
    <COMPONENT-REF DEST="APPLICATION-SOFTWARE-COMPONENT-
TYPE"/>CrossControl/SoftwareComponents/SeatHeatingController</COMPONENT-REF>
    <EVENTS>
        <TIMING-EVENT>
            <SHORT-NAME>SeatHeatingControllerMainTrigger</SHORT-NAME>
            <START-ON-EVENT-REF DEST="RUNNABLE-
ENTITY"/>CrossControl/SoftwareComponents/SeatHeatingControllerInternalBhv/SeatHeatingControlle
rMain</START-ON-EVENT-REF>
            <PERIOD>0.5</PERIOD>
        </TIMING-EVENT>
        <OPERATION-INVOKED-EVENT>
            <SHORT-NAME>SetPositionInvokedEvent</SHORT-NAME>
            <START-ON-EVENT-REF DEST="RUNNABLE-
ENTITY"/>CrossControl/SoftwareComponents/SeatHeatingControllerInternalBhv/SeatHeatingControlle
rSetPosition</START-ON-EVENT-REF>
            <OPERATION-IREF>
                <P-PORT-PROTOTYPE-REF DEST="P-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeatingController/RegulatorPosition</P-PORT-
PROTOTYPE-REF>
                <OPERATION-PROTOTYPE-REF DEST="OPERATION-
PROTOTYPE"/>CrossControl/Interfaces/IRegulatorPosition/SetPosition</OPERATION-PROTOTYPE-REF>
            </OPERATION-IREF>
        </OPERATION-INVOKED-EVENT>
    </EVENTS>
    <INTER-RUNNABLE-VARIABLES>
        <INTER-RUNNABLE-VARIABLE>
            <SHORT-NAME>InterRunPositionVar</SHORT-NAME>

```

```

        <TYPE-TREF DEST="INTEGER-TYPE"/>CrossControl/Datatypes/Int32</TYPE-TREF>
        <COMMUNICATION-APPROACH>IMPLICIT</COMMUNICATION-APPROACH>
        <INIT-VALUE-REF DEST="INTEGER-
LITERAL"/>CrossControl/Interfaces/ConstInitInt32/InitInt32</INIT-VALUE-REF>
        </INTER-RUNNABLE-VARIABLE>
        </INTER-RUNNABLE-VARIABLES>
        <RUNNABLES>
        <RUNNABLE-ENTITY>
        <SHORT-NAME>SeatHeatingControllerSetPosition</SHORT-NAME>
        <CAN-BE-INVOKED-CONCURRENTLY>true</CAN-BE-INVOKED-CONCURRENTLY>
        <SYMBOL>SeatHeatingControllerSetPosition</SYMBOL>
        <WRITTEN-VARIABLE-REFS>
        <WRITTEN-VARIABLE-REF DEST="INTER-RUNNABLE-
VARIABLE"/>CrossControl/SoftwareComponents/SeatHeatingControllerInternalBhv/InterRunPositionVa
r</WRITTEN-VARIABLE-REF>
        </WRITTEN-VARIABLE-REFS>
        </RUNNABLE-ENTITY>
        <RUNNABLE-ENTITY>
        <SHORT-NAME>SeatHeatingControllerMain</SHORT-NAME>
        <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
        <DATA-READ-ACCESS>
        <DATA-READ-ACCESS>
        <SHORT-NAME>LeftSeatStatusReadAccess</SHORT-NAME>
        <DATA-ELEMENT-IREF>
        <R-PORT-PROTOTYPE-REF DEST="R-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeatingController/LeftSeatStatus</R-PORT-
PROTOTYPE-REF>
        <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-
PROTOTYPE"/>CrossControl/Interfaces/ISeatStatus/PassengerOnSeat</DATA-ELEMENT-PROTOTYPE-REF>
        </DATA-ELEMENT-IREF>
        </DATA-READ-ACCESS>
        <DATA-READ-ACCESS>
        <SHORT-NAME>RightSeatStatusReadAccess</SHORT-NAME>
        <DATA-ELEMENT-IREF>
        <R-PORT-PROTOTYPE-REF DEST="R-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeatingController/RightSeatStatus</R-PORT-
PROTOTYPE-REF>
        <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-
PROTOTYPE"/>CrossControl/Interfaces/ISeatStatus/PassengerOnSeat</DATA-ELEMENT-PROTOTYPE-REF>
        </DATA-ELEMENT-IREF>
        </DATA-READ-ACCESS>
        </DATA-READ-ACCESS>
        <READ-VARIABLE-REFS>
        <READ-VARIABLE-REF DEST="INTER-RUNNABLE-
VARIABLE"/>CrossControl/SoftwareComponents/SeatHeatingControllerInternalBhv/InterRunPositionVa
r</READ-VARIABLE-REF>
        </READ-VARIABLE-REFS>
        <SERVER-CALL-POINTS>
        <SYNCHRONOUS-SERVER-CALL-POINT>
        <SHORT-NAME>HeaterLevelsCallPoint</SHORT-NAME>
        <OPERATION-IREFS>
        <OPERATION-IREF>
        <R-PORT-PROTOTYPE-REF DEST="R-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeatingController/HeaterLevels</R-PORT-
PROTOTYPE-REF>
        <OPERATION-PROTOTYPE-REF DEST="OPERATION-
PROTOTYPE"/>CrossControl/Interfaces/IHeaterLevel/SetHeat</OPERATION-PROTOTYPE-REF>
        </OPERATION-IREF>
        </OPERATION-IREFS>
        </SYNCHRONOUS-SERVER-CALL-POINT>
        </SERVER-CALL-POINTS>
        <SYMBOL>SeatHeatingControllerMain</SYMBOL>
        </RUNNABLE-ENTITY>
        </RUNNABLES>
        <SUPPORTS-MULTIPLE-INSTANTIATION>false</SUPPORTS-MULTIPLE-INSTANTIATION>
        </INTERNAL-BEHAVIOR>
        <SWC-IMPLEMENTATION S="">
        <SHORT-NAME>SeatHeatingControllerImpl</SHORT-NAME>
        <BEHAVIOR-REF DEST="INTERNAL-
BEHAVIOR"/>CrossControl/SoftwareComponents/SeatHeatingControllerInternalBhv</BEHAVIOR-REF>
        </SWC-IMPLEMENTATION>
        </ELEMENTS>
        </AR-PACKAGE>
        </SUB-PACKAGES>
        </AR-PACKAGE>
    </TOP-LEVEL-PACKAGES></AUTOSAR>

```

## 11.4 SeatSensor.arxml

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/3.1.4"><TOP-LEVEL-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>CrossControl</SHORT-NAME>
    <SUB-PACKAGES>
      <AR-PACKAGE>
        <SHORT-NAME>SoftwareComponents</SHORT-NAME>
        <ELEMENTS>
          <APPLICATION-SOFTWARE-COMPONENT-TYPE>
            <SHORT-NAME>SeatSensor</SHORT-NAME>
            <PORTS>
              <P-PORT-PROTOTYPE>
                <SHORT-NAME>Status</SHORT-NAME>
                <PROVIDED-COM-SPECS>
                  <UNQUEUED-SENDER-COM-SPEC>
                    <DATA-ELEMENT-REF DEST="DATA-ELEMENT-
PROTOTYPE"/>CrossControl/Interfaces/ISeatStatus/PassengerOnSeat</DATA-ELEMENT-REF>
                    <INIT-VALUE-REF DEST="BOOLEAN-
LITERAL"/>CrossControl/Interfaces/ConstInitBoolean/InitBoolean</INIT-VALUE-REF>
                  </UNQUEUED-SENDER-COM-SPEC>
                </PROVIDED-COM-SPECS>
                <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-
INTERFACE"/>CrossControl/Interfaces/ISeatStatus</PROVIDED-INTERFACE-TREF>
              </P-PORT-PROTOTYPE>
              <R-PORT-PROTOTYPE>
                <SHORT-NAME>SensorIO</SHORT-NAME>
                <REQUIRED-COM-SPECS>
                  <UNQUEUED-RECEIVER-COM-SPEC>
                    <DATA-ELEMENT-REF DEST="DATA-ELEMENT-
PROTOTYPE"/>CrossControl/Interfaces/ISensorIO/SensorValue</DATA-ELEMENT-REF>
                    <INIT-VALUE-REF DEST="BOOLEAN-
LITERAL"/>CrossControl/Interfaces/ConstInitBoolean/InitBoolean</INIT-VALUE-REF>
                  </UNQUEUED-RECEIVER-COM-SPEC>
                </REQUIRED-COM-SPECS>
                <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-
INTERFACE"/>CrossControl/Interfaces/ISensorIO</REQUIRED-INTERFACE-TREF>
              </R-PORT-PROTOTYPE>
            </PORTS>
          </APPLICATION-SOFTWARE-COMPONENT-TYPE>
          <SWC-IMPLEMENTATION>
            <SHORT-NAME>SeatSensorImpl</SHORT-NAME>
            <BEHAVIOR-REF DEST="INTERNAL-
BEHAVIOR"/>CrossControl/SoftwareComponents/SeatSensorInternalBhv</BEHAVIOR-REF>
          </SWC-IMPLEMENTATION>
          <INTERNAL-BEHAVIOR>
            <SHORT-NAME>SeatSensorInternalBhv</SHORT-NAME>
            <COMPONENT-REF DEST="APPLICATION-SOFTWARE-COMPONENT-
TYPE"/>CrossControl/SoftwareComponents/SeatSensor</COMPONENT-REF>
            <EVENTS>
              <TIMING-EVENT>
                <SHORT-NAME>SeatSensorMainTrigger</SHORT-NAME>
                <START-ON-EVENT-REF DEST="RUNNABLE-
ENTITY"/>CrossControl/SoftwareComponents/SeatSensorInternalBhv/SeatSensorMain</START-ON-EVENT-
REF>
                <PERIOD>1.0</PERIOD>
              </TIMING-EVENT>
            </EVENTS>
            <RUNNABLES>
              <RUNNABLE-ENTITY>
                <SHORT-NAME>SeatSensorMain</SHORT-NAME>
                <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
                <DATA-READ-ACCESS>
                  <DATA-READ-ACCESS>
                    <SHORT-NAME>SensorIOReadAccess</SHORT-NAME>
                    <DATA-ELEMENT-IREF>
                      <R-PORT-PROTOTYPE-REF DEST="R-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatSensor/SensorIO</R-PORT-PROTOTYPE-REF>
                    <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-
PROTOTYPE"/>CrossControl/Interfaces/ISensorIO/SensorValue</DATA-ELEMENT-PROTOTYPE-REF>
                  </DATA-ELEMENT-IREF>
                </DATA-READ-ACCESS>
                </DATA-READ-ACCESS>
                <DATA-WRITE-ACCESS>
                  <DATA-WRITE-ACCESS>
                    <SHORT-NAME>StatusWriteAccess</SHORT-NAME>
```



```

        <DATA-ELEMENT-IREF>
        <P-PORT-PROTOTYPE-REF DEST="P-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatSensor/Status</P-PORT-PROTOTYPE-REF>
        <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-
PROTOTYPE"/>CrossControl/Interfaces/ISeatStatus/PassengerOnSeat</DATA-ELEMENT-PROTOTYPE-REF>
        </DATA-ELEMENT-IREF>
    </DATA-WRITE-ACCESS>
</DATA-WRITE-ACCESS>
    <SYMBOL>SeatSensorMain</SYMBOL>
</RUNNABLE-ENTITY>
</RUNNABLES>
    <SUPPORTS-MULTIPLE-INSTITIATION>true</SUPPORTS-MULTIPLE-INSTITIATION>
</INTERNAL-BEHAVIOR>
</ELEMENTS>
</AR-PACKAGE>
</SUB-PACKAGES>
</AR-PACKAGE>
</TOP-LEVEL-PACKAGES></AUTOSAR>

```

## 11.5 DatatypesAndInterfaces.arxml

```

<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/3.1.4"><TOP-LEVEL-PACKAGES>
    <AR-PACKAGE>
        <SHORT-NAME>CrossControl</SHORT-NAME>
        <SUB-PACKAGES>
            <AR-PACKAGE>
                <SHORT-NAME>Datatypes</SHORT-NAME>
                <ELEMENTS>
                    <INTEGER-TYPE>
                        <SHORT-NAME>Int32</SHORT-NAME>
                        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">-2147483648</LOWER-LIMIT>
                        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">2147483647</UPPER-LIMIT>
                    </INTEGER-TYPE>
                    <BOOLEAN-TYPE>
                        <SHORT-NAME>Boolean</SHORT-NAME>
                    </BOOLEAN-TYPE>
                </ELEMENTS>
            </AR-PACKAGE>
        </SUB-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>Interfaces</SHORT-NAME>
            <ELEMENTS>
                <CONSTANT-SPECIFICATION>
                    <SHORT-NAME>ConstInitInt32</SHORT-NAME>
                    <VALUE>
                        <INTEGER-LITERAL>
                            <SHORT-NAME>InitInt32</SHORT-NAME>
                            <TYPE-TREF DEST="INTEGER-TYPE"/>CrossControl/Datatypes/Int32</TYPE-TREF>
                            <VALUE>0</VALUE>
                        </INTEGER-LITERAL>
                    </VALUE>
                </CONSTANT-SPECIFICATION>
                <CONSTANT-SPECIFICATION>
                    <SHORT-NAME>ConstInitBoolean</SHORT-NAME>
                    <VALUE>
                        <BOOLEAN-LITERAL>
                            <SHORT-NAME>InitBoolean</SHORT-NAME>
                            <TYPE-TREF DEST="BOOLEAN-TYPE"/>CrossControl/Datatypes/Boolean</TYPE-TREF>
                            <VALUE>>false</VALUE>
                        </BOOLEAN-LITERAL>
                    </VALUE>
                </CONSTANT-SPECIFICATION>
                <CLIENT-SERVER-INTERFACE>
                    <SHORT-NAME>IRegulatorPosition</SHORT-NAME>
                    <OPERATIONS>
                        <OPERATION-PROTOTYPE>
                            <SHORT-NAME>SetPosition</SHORT-NAME>
                            <ARGUMENTS>
                                <ARGUMENT-PROTOTYPE>
                                    <SHORT-NAME>Position</SHORT-NAME>
                                    <TYPE-TREF DEST="INTEGER-TYPE"/>CrossControl/Datatypes/Int32</TYPE-TREF>
                                    <DIRECTION>IN</DIRECTION>
                                </ARGUMENT-PROTOTYPE>
                            </ARGUMENTS>
                            <POSSIBLE-ERROR-REFS>

```

```

        <POSSIBLE-ERROR-REF DEST="APPLICATION-
ERROR"/>CrossControl/Interfaces/IRegulatorPosition/RegulatorHardwareError</POSSIBLE-ERROR-REF>
        <POSSIBLE-ERROR-REF DEST="APPLICATION-
ERROR"/>CrossControl/Interfaces/IRegulatorPosition/RegulatorSoftwareError</POSSIBLE-ERROR-REF>
    </POSSIBLE-ERROR-REFS>
    </OPERATION-PROTOTYPE>
</OPERATIONS>
<POSSIBLE-ERRORS>
    <APPLICATION-ERROR>
        <SHORT-NAME>RegulatorHardwareError</SHORT-NAME>
        <ERROR-CODE>-1</ERROR-CODE>
    </APPLICATION-ERROR>
    <APPLICATION-ERROR>
        <SHORT-NAME>RegulatorSoftwareError</SHORT-NAME>
        <ERROR-CODE>-2</ERROR-CODE>
    </APPLICATION-ERROR>
</POSSIBLE-ERRORS>
</CLIENT-SERVER-INTERFACE>
<CLIENT-SERVER-INTERFACE>
    <SHORT-NAME>IHeaterLevel</SHORT-NAME>
    <OPERATIONS>
        <OPERATION-PROTOTYPE>
            <SHORT-NAME>SetHeat</SHORT-NAME>
            <ARGUMENTS>
                <ARGUMENT-PROTOTYPE>
                    <SHORT-NAME>LeftHeatLevel</SHORT-NAME>
                    <TYPE-TREF DEST="INTEGER-TYPE"/>CrossControl/Datatypes/Int32</TYPE-TREF>
                    <DIRECTION>IN</DIRECTION>
                </ARGUMENT-PROTOTYPE>
                <ARGUMENT-PROTOTYPE>
                    <SHORT-NAME>RightHeatLevel</SHORT-NAME>
                    <TYPE-TREF DEST="INTEGER-TYPE"/>CrossControl/Datatypes/Int32</TYPE-TREF>
                    <DIRECTION>IN</DIRECTION>
                </ARGUMENT-PROTOTYPE>
            </ARGUMENTS>
            <POSSIBLE-ERROR-REFS>
                <POSSIBLE-ERROR-REF DEST="APPLICATION-
ERROR"/>CrossControl/Interfaces/IHeaterLevel/HeaterHardwareError</POSSIBLE-ERROR-REF>
                <POSSIBLE-ERROR-REF DEST="APPLICATION-
ERROR"/>CrossControl/Interfaces/IHeaterLevel/HeaterSoftwareError</POSSIBLE-ERROR-REF>
            </POSSIBLE-ERROR-REFS>
        </OPERATION-PROTOTYPE>
    </OPERATIONS>
    <POSSIBLE-ERRORS>
        <APPLICATION-ERROR>
            <SHORT-NAME>HeaterSoftwareError</SHORT-NAME>
            <ERROR-CODE>-2</ERROR-CODE>
        </APPLICATION-ERROR>
        <APPLICATION-ERROR>
            <SHORT-NAME>HeaterHardwareError</SHORT-NAME>
            <ERROR-CODE>-1</ERROR-CODE>
        </APPLICATION-ERROR>
    </POSSIBLE-ERRORS>
</CLIENT-SERVER-INTERFACE>
<SENDER-RECEIVER-INTERFACE>
    <SHORT-NAME>ISeatStatus</SHORT-NAME>
    <DATA-ELEMENTS>
        <DATA-ELEMENT-PROTOTYPE S="">
            <SHORT-NAME>PassengerOnSeat</SHORT-NAME>
            <TYPE-TREF DEST="BOOLEAN-TYPE"/>CrossControl/Datatypes/Boolean</TYPE-TREF>
            <IS-QUEUED>>false</IS-QUEUED>
        </DATA-ELEMENT-PROTOTYPE>
    </DATA-ELEMENTS>
</SENDER-RECEIVER-INTERFACE>
<SENDER-RECEIVER-INTERFACE>
    <SHORT-NAME>IRegulatorIO</SHORT-NAME>
    <DATA-ELEMENTS>
        <DATA-ELEMENT-PROTOTYPE>
            <SHORT-NAME>RegulatorValue</SHORT-NAME>
            <TYPE-TREF DEST="INTEGER-TYPE"/>CrossControl/Datatypes/Int32</TYPE-TREF>
            <IS-QUEUED>>false</IS-QUEUED>
        </DATA-ELEMENT-PROTOTYPE>
    </DATA-ELEMENTS>
</SENDER-RECEIVER-INTERFACE>
<SENDER-RECEIVER-INTERFACE>
    <SHORT-NAME>IHeaterIO</SHORT-NAME>
    <DATA-ELEMENTS>

```

```

        <DATA-ELEMENT-PROTOTYPE>
        <SHORT-NAME>HeaterValue</SHORT-NAME>
        <TYPE-TREF DEST="INTEGER-TYPE"/>/CrossControl/Datatypes/Int32</TYPE-TREF>
        <IS-QUEUED>>false</IS-QUEUED>
        </DATA-ELEMENT-PROTOTYPE>
    </DATA-ELEMENTS>
</SENDER-RECEIVER-INTERFACE>
<SENDER-RECEIVER-INTERFACE>
    <SHORT-NAME>ISensorIO</SHORT-NAME>
    <DATA-ELEMENTS>
        <DATA-ELEMENT-PROTOTYPE>
        <SHORT-NAME>SensorValue</SHORT-NAME>
        <TYPE-TREF DEST="BOOLEAN-TYPE"/>/CrossControl/Datatypes/Boolean</TYPE-TREF>
        <IS-QUEUED>>false</IS-QUEUED>
        </DATA-ELEMENT-PROTOTYPE>
    </DATA-ELEMENTS>
</SENDER-RECEIVER-INTERFACE>
</ELEMENTS>
</AR-PACKAGE>
</SUB-PACKAGES>
</AR-PACKAGE>
</TOP-LEVEL-PACKAGES></AUTOSAR>

```

## 11.6 SeatHeaterComposition.arxml

```

<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/3.1.4"><TOP-LEVEL-PACKAGES>
    <AR-PACKAGE>
        <SHORT-NAME>CrossControl</SHORT-NAME>
        <SUB-PACKAGES>
            <AR-PACKAGE>
                <SHORT-NAME>SoftwareComponents</SHORT-NAME>
                <ELEMENTS>
                    <COMPOSITION-TYPE>
                        <SHORT-NAME>SeatHeaterComposition</SHORT-NAME>
                        <COMPONENTS>
                            <COMPONENT-PROTOTYPE>
                                <SHORT-NAME>SeatHeatingController0</SHORT-NAME>
                                <TYPE-TREF DEST="APPLICATION-SOFTWARE-COMPONENT-TYPE"/>/CrossControl/SoftwareComponents/SeatHeatingController</TYPE-TREF>
                                </COMPONENT-PROTOTYPE>
                                <COMPONENT-PROTOTYPE>
                                    <SHORT-NAME>SeatHeater0</SHORT-NAME>
                                    <TYPE-TREF DEST="APPLICATION-SOFTWARE-COMPONENT-TYPE"/>/CrossControl/SoftwareComponents/SeatHeater</TYPE-TREF>
                                    </COMPONENT-PROTOTYPE>
                                    <COMPONENT-PROTOTYPE>
                                        <SHORT-NAME>HeatRegulator0</SHORT-NAME>
                                        <TYPE-TREF DEST="APPLICATION-SOFTWARE-COMPONENT-TYPE"/>/CrossControl/SoftwareComponents/HeatRegulator</TYPE-TREF>
                                        </COMPONENT-PROTOTYPE>
                                        <COMPONENT-PROTOTYPE>
                                            <SHORT-NAME>SeatSensor0</SHORT-NAME>
                                            <TYPE-TREF DEST="APPLICATION-SOFTWARE-COMPONENT-TYPE"/>/CrossControl/SoftwareComponents/SeatSensor</TYPE-TREF>
                                            </COMPONENT-PROTOTYPE>
                                            <COMPONENT-PROTOTYPE>
                                                <SHORT-NAME>SeatSensor1</SHORT-NAME>
                                                <TYPE-TREF DEST="APPLICATION-SOFTWARE-COMPONENT-TYPE"/>/CrossControl/SoftwareComponents/SeatSensor</TYPE-TREF>
                                                </COMPONENT-PROTOTYPE>
                                            </COMPONENTS>
                                        <CONNECTORS>
                                            <ASSEMBLY-CONNECTOR-PROTOTYPE>
                                                <SHORT-NAME>RegulatorPositionConnector</SHORT-NAME>
                                                <PROVIDER-IREF>
                                                    <COMPONENT-PROTOTYPE-REF DEST="COMPONENT-PROTOTYPE"/>/CrossControl/SoftwareComponents/SeatHeaterComposition/SeatHeatingController0</COMPONENT-PROTOTYPE-REF>
                                                    <P-PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE"/>/CrossControl/SoftwareComponents/SeatHeatingController/RegulatorPosition</P-PORT-PROTOTYPE-REF>
                                                </PROVIDER-IREF>
                                                <REQUESTER-IREF>

```

```

        <COMPONENT-PROTOTYPE-REF DEST="COMPONENT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeaterComposition/HeatRegulator0</COMPONENT-
PROTOTYPE-REF>
        <R-PORT-PROTOTYPE-REF DEST="R-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/HeatRegulator/Position</R-PORT-PROTOTYPE-REF>
        </REQUESTER-IREF>
        </ASSEMBLY-CONNECTOR-PROTOTYPE>
        <ASSEMBLY-CONNECTOR-PROTOTYPE>
        <SHORT-NAME>LeftSeatStatusConnector</SHORT-NAME>
        <PROVIDER-IREF>
        <COMPONENT-PROTOTYPE-REF DEST="COMPONENT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeaterComposition/SeatSensor0</COMPONENT-
PROTOTYPE-REF>
        <P-PORT-PROTOTYPE-REF DEST="P-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatSensor/Status</P-PORT-PROTOTYPE-REF>
        </PROVIDER-IREF>
        <REQUESTER-IREF>
        <COMPONENT-PROTOTYPE-REF DEST="COMPONENT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeaterComposition/SeatHeatingController0</COMP
ONENT-PROTOTYPE-REF>
        <R-PORT-PROTOTYPE-REF DEST="R-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeatingController/LeftSeatStatus</R-PORT-
PROTOTYPE-REF>
        </REQUESTER-IREF>
        </ASSEMBLY-CONNECTOR-PROTOTYPE>
        <ASSEMBLY-CONNECTOR-PROTOTYPE>
        <SHORT-NAME>RightSeatStatusConnector</SHORT-NAME>
        <PROVIDER-IREF>
        <COMPONENT-PROTOTYPE-REF DEST="COMPONENT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeaterComposition/SeatSensor1</COMPONENT-
PROTOTYPE-REF>
        <P-PORT-PROTOTYPE-REF DEST="P-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatSensor/Status</P-PORT-PROTOTYPE-REF>
        </PROVIDER-IREF>
        <REQUESTER-IREF>
        <COMPONENT-PROTOTYPE-REF DEST="COMPONENT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeaterComposition/SeatHeatingController0</COMP
ONENT-PROTOTYPE-REF>
        <R-PORT-PROTOTYPE-REF DEST="R-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeatingController/RightSeatStatus</R-PORT-
PROTOTYPE-REF>
        </REQUESTER-IREF>
        </ASSEMBLY-CONNECTOR-PROTOTYPE>
        <ASSEMBLY-CONNECTOR-PROTOTYPE>
        <SHORT-NAME>HeaterLevelConnector</SHORT-NAME>
        <PROVIDER-IREF>
        <COMPONENT-PROTOTYPE-REF DEST="COMPONENT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeaterComposition/SeatHeater0</COMPONENT-
PROTOTYPE-REF>
        <P-PORT-PROTOTYPE-REF DEST="P-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeater/Levels</P-PORT-PROTOTYPE-REF>
        </PROVIDER-IREF>
        <REQUESTER-IREF>
        <COMPONENT-PROTOTYPE-REF DEST="COMPONENT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeaterComposition/SeatHeatingController0</COMP
ONENT-PROTOTYPE-REF>
        <R-PORT-PROTOTYPE-REF DEST="R-PORT-
PROTOTYPE"/>CrossControl/SoftwareComponents/SeatHeatingController/HeaterLevels</R-PORT-
PROTOTYPE-REF>
        </REQUESTER-IREF>
        </ASSEMBLY-CONNECTOR-PROTOTYPE>
        </CONNECTORS>
        </COMPOSITION-TYPE>
        </ELEMENTS>
        </AR-PACKAGE>
        </SUB-PACKAGES>
        </AR-PACKAGE>
    </TOP-LEVEL-PACKAGES></AUTOSAR>

```

## 12 Appendix 2: Demonstrator software components implementations

### 12.1 HeatRegulator.c

```
/*
*****
*/
FUNCTION DEFINITION
/*
*****
*/

/*
*****
* Function Name      : HeatRegulator_HeatRegulatorMain
* Description        : This service depicts a runnable entity on the application
*                     layer and called from RTE.
*
* Parameters         : None
*
* Called Functions   : Rte_Call_HeatRegulator_Position_SetPosition()
*
Rte_IRead_HeatRegulatorMain_RegulatorIO_RegulatorValue()
*
* Return Value       : None
*
*****
*/

void HeatRegulator_HeatRegulatorMain(void)
{
    Int32 regulatorPosition;
    Std_ReturnType retval;

    #if defined(RTE_TRACE_LOGGER_ENABLED)
        char sContent[512];

        // COQOS-tC: TraceEventRteRunnableSimple indicating that a thread enters the runnable
        'HeatRegulatorMain'
        snprintf(sContent, 512, "{\"type\":\"RteRunnable\",
        \"isReturn\":false,\"origin\":\"Runnable\",
        \"runnableName\":\"HeatRegulatorMain\", \"componentName\":\"HeatRegulator\",
        \"componentInstance\":\"%s\"}", Rte_Inst_HeatRegulator.InstanceName);
        traceEvent(sContent);

        // COQOS-tC: TraceEventRteRead indicating that the runnable calls Rte_IRead
        snprintf(sContent, 512, "{\"type\":\"RteIRead\",
        \"isReturn\":false,\"origin\":\"Runnable\",
        \"componentName\":\"HeatRegulator\", \"componentInstance\":\"%s\",
        \"portName\":\"RegulatorIO\", \"dataElementName\":\"RegulatorValue\"}",
        Rte_Inst_HeatRegulator.InstanceName);
        traceEvent(sContent);
    #endif

    regulatorPosition = Rte_IRead_HeatRegulatorMain_RegulatorIO_RegulatorValue();

    #if defined(RTE_TRACE_LOGGER_ENABLED)
        // COQOS-tC: TraceEventRteRead indicating that the runnable returned from calling
        Rte_IRead
        snprintf(sContent, 512, "{\"type\":\"RteIRead\",
        \"isReturn\":true,\"origin\":\"Runnable\",
        \"componentName\":\"HeatRegulator\", \"componentInstance\":\"%s\",
        \"portName\":\"RegulatorIO\", \"dataElementName\":\"RegulatorValue\", \"dataValue\":%d}
        ", Rte_Inst_HeatRegulator.InstanceName, regulatorPosition);
        traceEvent(sContent);

        // COQOS-tC: TraceEventRteRead indicating that the runnable calls Rte_Call
        snprintf(sContent, 512, "{\"type\":\"RteCall\",
        \"isReturn\":false,\"origin\":\"Runnable\",
        \"componentName\":\"HeatRegulator\", \"componentInstance\":\"%s\",
        \"portName\":\"Position\", \"operatorName\":\"SetPosition\"}",
        Rte_Inst_HeatRegulator.InstanceName);
        traceEvent(sContent);
    #endif

    #ifdef DEBUG_LOG
        Logger_log(LOGGER_SEVERITY_INFO, "[SWC ] HeatRegulatorMain()");
        Logger_log(LOGGER_SEVERITY_INFO, "[RTE ]
        Rte_IRead_HeatRegulatorMain_RegulatorIO_RegulatorValue(): %d", regulatorPosition);
        Logger_log(LOGGER_SEVERITY_INFO, "[RTE ] Rte_Call_Position_SetPosition( %d )\n",
        regulatorPosition);
    #endif
}
```

```

    retval = Rte_Call_Position_SetPosition(regulatorPosition);

    #if defined(RTE_TRACE_LOGGER_ENABLED)
        // COQOS-tC: TraceEventRteRead indicating that the runnable returned from calling
        Rte_Call
        snprintf(sContent, 512, "{\"type\":\"RteCall\",
        \"isReturn\":true,\"origin\":\"Runnable\",
        \"componentName\":\"HeatRegulator\", \"componentInstance\":\"%s\",
        \"portName\":\"Position\", \"operatorName\":\"SetPosition\", \"returnValue\":%d}",
        Rte_Inst_HeatRegulator.InstanceName, retval);
        traceEvent(sContent);

        // COQOS-tC: TraceEventRteRunnableSimple indicating that the thread leaves the
        runnable 'HeatRegulatorMain'
        snprintf(sContent, 512, "{\"type\":\"RteRunnable\",
        \"isReturn\":true,\"origin\":\"Runnable\",
        \"runnableName\":\"HeatRegulator\", \"componentName\":\"HeatRegulator\",
        \"componentInstance\":\"%s\"}",
        Rte_Inst_HeatRegulator.InstanceName); traceEvent(sContent);
    #endif
}

```

## 12.2 SeatHeater.c

```

/*****
/*          FUNCTION DEFINITION          */
*****/

/*****
* Function Name      : SeatHeater_SeatHeaterMain
* Description        : This service is called by the SeatHeatingController.
*
* Parameters         : leftHeatLevel[in]: Heat level to set for left seat
*                                     rightHeatLevel[in]: Heat
level to set for right seat
*
* Called Functions   : Rte_Write_LeftSeatHeaterIO_HeaterValue()
*
Rte_Write_RightSeatHeaterIO_HeaterValue()
*
* Return Value       : None
*
*****/

void SeatHeater_SeatHeaterMain(Int32 leftHeatLevel, Int32 rightHeatLevel)
{
    Std_ReturnType retval;

    #if defined(RTE_TRACE_LOGGER_ENABLED)
        char sContent[512];

        // COQOS-tC: TraceEventRteRunnableSimple indicating that a thread enters the runnable
        'SeatHeaterMain'
        snprintf(sContent, 512, "{\"type\":\"RteRunnable\",
        \"isReturn\":false,\"origin\":\"Runnable\",
        \"runnableName\":\"SeatHeaterMain\", \"componentName\":\"SeatHeater\",
        \"componentInstance\":\"%s\"}", Rte_Inst_SeatHeater.InstanceName);
        traceEvent(sContent);

        // COQOS-tC: TraceEventRteRead indicating that the runnable calls Rte_IWrite
        snprintf(sContent, 512, "{\"type\":\"RteIWrite\",
        \"isReturn\":false,\"origin\":\"Runnable\",
        \"componentName\":\"SeatHeater\", \"componentInstance\":\"%s\",
        \"portName\":\"LeftSeatHeaterIO\", \"dataElementName\":\"HeaterValue\"}",
        Rte_Inst_SeatHeater.InstanceName);
        traceEvent(sContent);
    #endif

    Rte_IWrite_SeatHeaterMain_LeftSeatHeaterIO_HeaterValue(leftHeatLevel);

    #if defined(RTE_TRACE_LOGGER_ENABLED)
        // COQOS-tC: TraceEventRteRead indicating that the runnable returned from calling
        Rte_IWrite
        snprintf(sContent, 512, "{\"type\":\"RteIWrite\",
        \"isReturn\":true,\"origin\":\"Runnable\",

```

```

        \"componentName\\\": \"SeatHeater\\\", \"componentInstance\\\": \"%s\\\",
        \"portName\\\": \"LeftSeatHeaterIO\\\", \"dataElementName\\\": \"HeaterValue\\\"}\",
        Rte_Inst_SeatHeater.InstanceName);
        traceEvent(sContent);

        // COQOS-tC: TraceEventRteRead indicating that the runnable calls Rte_IWrite
        snprintf(sContent, 512, \"{\\\"type\\\": \\\"RteIWrite\\\",
        \\\"isReturn\\\": false, \\\"origin\\\": \\\"Runnable\\\",
        \\\"componentName\\\": \\\"SeatHeater\\\", \"componentInstance\\\": \"%s\\\",
        \\\"portName\\\": \\\"RightSeatHeaterIO\\\", \"dataElementName\\\": \\\"HeaterValue\\\"}\",
        Rte_Inst_SeatHeater.InstanceName);
        traceEvent(sContent);
    #endif

    Rte_IWrite_SeatHeaterMain_RightSeatHeaterIO_HeaterValue(rightHeatLevel);

    #if defined(RTE_TRACE_LOGGER_ENABLED)
        // COQOS-tC: TraceEventRteRead indicating that the runnable returned from calling
        Rte_IWrite
        snprintf(sContent, 512, \"{\\\"type\\\": \\\"RteIWrite\\\",
        \\\"isReturn\\\": true, \\\"origin\\\": \\\"Runnable\\\",
        \\\"componentName\\\": \\\"SeatHeater\\\", \"componentInstance\\\": \"%s\\\",
        \\\"portName\\\": \\\"RightSeatHeaterIO\\\", \"dataElementName\\\": \\\"HeaterValue\\\"}\",
        Rte_Inst_SeatHeater.InstanceName);
        traceEvent(sContent);
    #endif

    #ifdef DEBUG_LOG
        Logger_log(LOGGER_SEVERITY_INFO, \"[SWC ] SeatHeaterMain()\");
        Logger_log(LOGGER_SEVERITY_INFO, \"[RTE ]
        Rte_IWrite_SeatHeaterMain_LeftSeatHeaterIO_HeaterValue(): %d\", leftHeatLevel);
        Logger_log(LOGGER_SEVERITY_INFO, \"[RTE ]
        Rte_IWrite_SeatHeaterMain_RightSeatHeaterIO_HeaterValue(): %d\\n\", rightHeatLevel);
    #endif

    #if defined(RTE_TRACE_LOGGER_ENABLED)
        // COQOS-tC: TraceEventRteRunnableSimple indicating that the thread leaves the
        runnable 'SeatHeaterMain'
        snprintf(sContent, 512, \"{\\\"type\\\": \\\"RteRunnable\\\",
        \\\"isReturn\\\": true, \\\"origin\\\": \\\"Runnable\\\",
        \\\"runnableName\\\": \\\"SeatHeaterMain\\\", \"componentName\\\": \\\"SeatHeater\\\",
        \\\"componentInstance\\\": \"%s\\\"}\", Rte_Inst_SeatHeater.InstanceName);
        traceEvent(sContent);
    #endif
}

```

## 12.3 SeatHeatingController.c

```

/*****
*
* FUNCTION DEFINITION
*
*****/

/*****
* Function Name      : SeatHeatingController_SeatHeatingControllerMain
* Description        : This service depicts a runnable entity on the application
*                     layer and called from RTE.
*
* Parameters         : None
*
* Called Functions   : Rte_Call_SeatHeatingController_HeaterLevels_SetHeat()
*
Rte_IRead_SeatHeatingControllerMain_LeftSeatStatus_PassengerOnSeat()
*
Rte_IRead_SeatHeatingControllerMain_RightSeatStatus_PassengerOnSeat()
*
Rte_IrvIRead_SeatHeatingControllerMain_InterRunPositionVar()
*
* Return Value       : None
*****/

void SeatHeatingController_SeatHeatingControllerMain(void)
{
    Boolean leftPassengerOnSeat;
    Boolean rightPassengerOnSeat;
    Int32 leftHeatLevel;
    Int32 regulatorPosition;

```

```

Int32 rightHeatLevel;
Std_ReturnType retVal;

#if defined(RTE_TRACE_LOGGER_ENABLED)
    char sContent[512];

    // COQOS-tC: TraceEventRteRunnableSimple indicating that a thread enters the runnable
    'SeatHeatingControllerMain'
    snprintf(sContent, 512, "{\"type\":\"RteRunnable\",
    \"isReturn\":false,\"origin\":\"Runnable\",
    \"runnableName\":\"SeatHeatingControllerMain\", \"componentName\":\"SeatHeatingControl
    ler\", \"componentInstance\":\"%s\"}", Rte_Inst_SeatHeatingController.InstanceName);
    traceEvent(sContent);

    // COQOS-tC: TraceEventRteRead indicating that the runnable calls Rte_IRead
    snprintf(sContent, 512, "{\"type\":\"RteIRead\",
    \"isReturn\":false,\"origin\":\"Runnable\",
    \"componentName\":\"SeatHeatingController\", \"componentInstance\":\"%s\",
    \"portName\":\"LeftSeatStatus\", \"dataElementName\":\"PassengerOnSeat\"}",
    Rte_Inst_SeatHeatingController.InstanceName);
    traceEvent(sContent);
#endif

leftPassengerOnSeat =
Rte_IRead_SeatHeatingControllerMain_LeftSeatStatus_PassengerOnSeat();

#if defined(RTE_TRACE_LOGGER_ENABLED)
    // COQOS-tC: TraceEventRteRead indicating that the runnable returned from calling
    Rte_IRead
    snprintf(sContent, 512, "{\"type\":\"RteIRead\",
    \"isReturn\":true,\"origin\":\"Runnable\",
    \"componentName\":\"SeatHeatingController\", \"componentInstance\":\"%s\",
    \"portName\":\"LeftSeatStatus\", \"dataElementName\":\"PassengerOnSeat\", \"dataValue\"
    :%d}", Rte_Inst_SeatHeatingController.InstanceName, leftPassengerOnSeat);
    traceEvent(sContent);

    // COQOS-tC: TraceEventRteRead indicating that the runnable calls Rte_IRead
    snprintf(sContent, 512, "{\"type\":\"RteIRead\",
    \"isReturn\":false,\"origin\":\"Runnable\",
    \"componentName\":\"SeatHeatingController\", \"componentInstance\":\"%s\",
    \"portName\":\"RightSeatStatus\", \"dataElementName\":\"PassengerOnSeat\"}",
    Rte_Inst_SeatHeatingController.InstanceName);
    traceEvent(sContent);
#endif

rightPassengerOnSeat =
Rte_IRead_SeatHeatingControllerMain_RightSeatStatus_PassengerOnSeat();

#if defined(RTE_TRACE_LOGGER_ENABLED)
    // COQOS-tC: TraceEventRteRead indicating that the runnable returned from calling
    Rte_IRead
    snprintf(sContent, 512, "{\"type\":\"RteIRead\",
    \"isReturn\":true,\"origin\":\"Runnable\",
    \"componentName\":\"SeatHeatingController\", \"componentInstance\":\"%s\",
    \"portName\":\"RightSeatStatus\", \"dataElementName\":\"PassengerOnSeat\", \"dataValue\"
    :%d}", Rte_Inst_SeatHeatingController.InstanceName, leftPassengerOnSeat);
    traceEvent(sContent);
#endif

regulatorPosition = Rte_IrvIRead_SeatHeatingControllerMain_InterRunPositionVar();

leftHeatLevel = regulatorPosition * leftPassengerOnSeat;
rightHeatLevel = regulatorPosition * rightPassengerOnSeat;

#if defined(RTE_TRACE_LOGGER_ENABLED)
    // COQOS-tC: TraceEventRteRead indicating that the runnable calls Rte_Call
    snprintf(sContent, 512, "{\"type\":\"RteCall\",
    \"isReturn\":false,\"origin\":\"Runnable\",
    \"componentName\":\"SeatHeatingController\", \"componentInstance\":\"%s\",
    \"portName\":\"HeaterLevels\", \"operatorName\":\"SetHeat\"}",
    Rte_Inst_SeatHeatingController.InstanceName);
    traceEvent(sContent);
#endif

#ifdef DEBUG_LOG
    Logger_log(LOGGER_SEVERITY_INFO, "[SWC ] SeatHeatingControllerMain()");

```





```

        // COQOS-tC: TraceEventRteRunnableSimple indicating that the thread leaves the
        runnable 'SeatHeatingControllerSetPosition'
        snprintf(sContent, 512, "{\"type\":\"RteRunnable\",
        \"isReturn\":true,\"origin\":\"Runnable\",
        \"runnableName\":\"SeatHeatingControllerSetPosition\", \"componentName\":\"SeatHeating
        Controller\", \"componentInstance\":\"%s\"}",
        Rte_Inst_SeatHeatingController.InstanceName);
        traceEvent(sContent);
    #endif
}

```

## 12.4 SeatSensor.c

```

/*****
/*          FUNCTION DEFINITION          */
*****/

/*****
* Function Name      : SeatSensor_SeatSensorMain
* Description        : This service depicts a runnable entity on the application
*                      layer and called from RTE.
*
* Parameters         : sensor[in]: Seat sensor
*
* Called Functions   : Rte_IRead_SeatSensorMain_SensorIO_SensorValue()
*
Rte_IWrite_SeatSensorMain_Status_PassengerOnSeat()
*
* Return Value       : None
*
*****/

void SeatSensor_SeatSensorMain(const Rte_CDS_SeatSensor* sensor)
{
    Boolean sensorStatus;

    #if defined(RTE_TRACE_LOGGER_ENABLED)
        char sContent[512];

        // COQOS-tC: TraceEventRteRunnableSimple indicating that a thread enters the runnable
        'SeatSensorMain'
        snprintf(sContent, 512, "{\"type\":\"RteRunnable\",
        \"isReturn\":false,\"origin\":\"Runnable\",
        \"runnableName\":\"SeatSensorMain\", \"componentName\":\"SeatSensor\",
        \"componentInstance\":\"%s\"}", sensor->InstanceName);
        traceEvent(sContent);

        // COQOS-tC: TraceEventRteRead indicating that the runnable calls Rte_IRead
        snprintf(sContent, 512, "{\"type\":\"RteIRead\",
        \"isReturn\":false,\"origin\":\"Runnable\",
        \"componentName\":\"SeatSensor\", \"componentInstance\":\"%s\",
        \"portName\":\"SensorIO\", \"dataElementName\":\"SensorValue\"}", sensor-
        >InstanceName);
        traceEvent(sContent);
    #endif

    sensorStatus = Rte_IRead_SeatSensorMain_SensorIO_SensorValue(sensor);

    #if defined(RTE_TRACE_LOGGER_ENABLED)
        // COQOS-tC: TraceEventRteRead indicating that the runnable returned from calling
        Rte_IRead
        snprintf(sContent, 512, "{\"type\":\"RteIRead\",
        \"isReturn\":true,\"origin\":\"Runnable\",
        \"componentName\":\"SeatSensor\", \"componentInstance\":\"%s\",
        \"portName\":\"SensorIO\", \"dataElementName\":\"SensorValue\", \"dataValue\":%d}",
        sensor->InstanceName, sensorStatus);
        traceEvent(sContent);

        // COQOS-tC: TraceEventRteRead indicating that the runnable calls Rte_IWrite
        snprintf(sContent, 512, "{\"type\":\"RteIWrite\",
        \"isReturn\":false,\"origin\":\"Runnable\",
        \"componentName\":\"SeatSensor\", \"componentInstance\":\"%s\",
        \"portName\":\"Status\", \"dataElementName\":\"PassengerOnSeat\"}", sensor-
        >InstanceName);
        traceEvent(sContent);
    #endif
}

```

```

Rte_IWrite_SeatSensorMain_Status_PassengerOnSeat(sensor, sensorStatus);

#if defined(RTE_TRACE_LOGGER_ENABLED)
    // COQOS-tC: TraceEventRteRead indicating that the runnable returned from calling
    Rte_IWrite
    snprintf(sContent, 512, "{\"type\":\"RteIWrite\",
    \"isReturn\":true,\"origin\":\"Runnable\",
    \"componentName\":\"SeatSensor\", \"componentInstance\":\"%s\",
    \"portName\":\"Status\", \"dataElementName\":\"PassengerOnSeat\"}", sensor-
    >InstanceName);
    traceEvent(sContent);
#endif

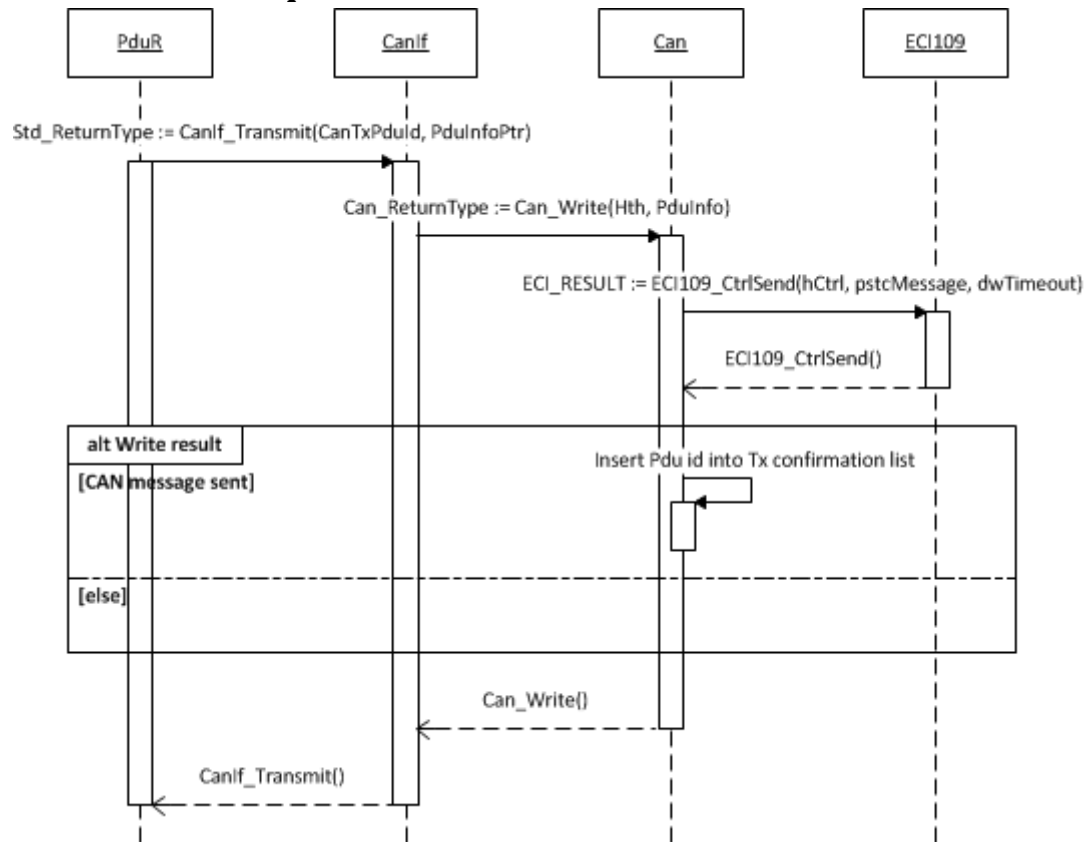
#ifdef DEBUG_LOG
    Logger_log(LOGGER_SEVERITY_INFO, "[SWC ] SeatSensorMain() ( %s ) ", sensor-
    >InstanceName);
    Logger_log(LOGGER_SEVERITY_INFO, "[RTE ]
    Rte_IRead_SeatSensorMain_SensorIO_SensorValue(): %d", sensorStatus);
    Logger_log(LOGGER_SEVERITY_INFO, "[RTE ]
    Rte_IWrite_SeatSensorMain_Status_PassengerOnSeat(): %d\n", sensorStatus);
#endif

#if defined(RTE_TRACE_LOGGER_ENABLED)
    // COQOS-tC: TraceEventRteRunnableSimple indicating that the thread leaves the
    runnable 'SeatSensorMain'
    snprintf(sContent, 512, "{\"type\":\"RteRunnable\",
    \"isReturn\":true,\"origin\":\"Runnable\",
    \"runnableName\":\"SeatSensorMain\", \"componentName\":\"SeatSensor\",
    \"componentInstance\":\"%s\"}", sensor->InstanceName);
    traceEvent(sContent);
#endif
}

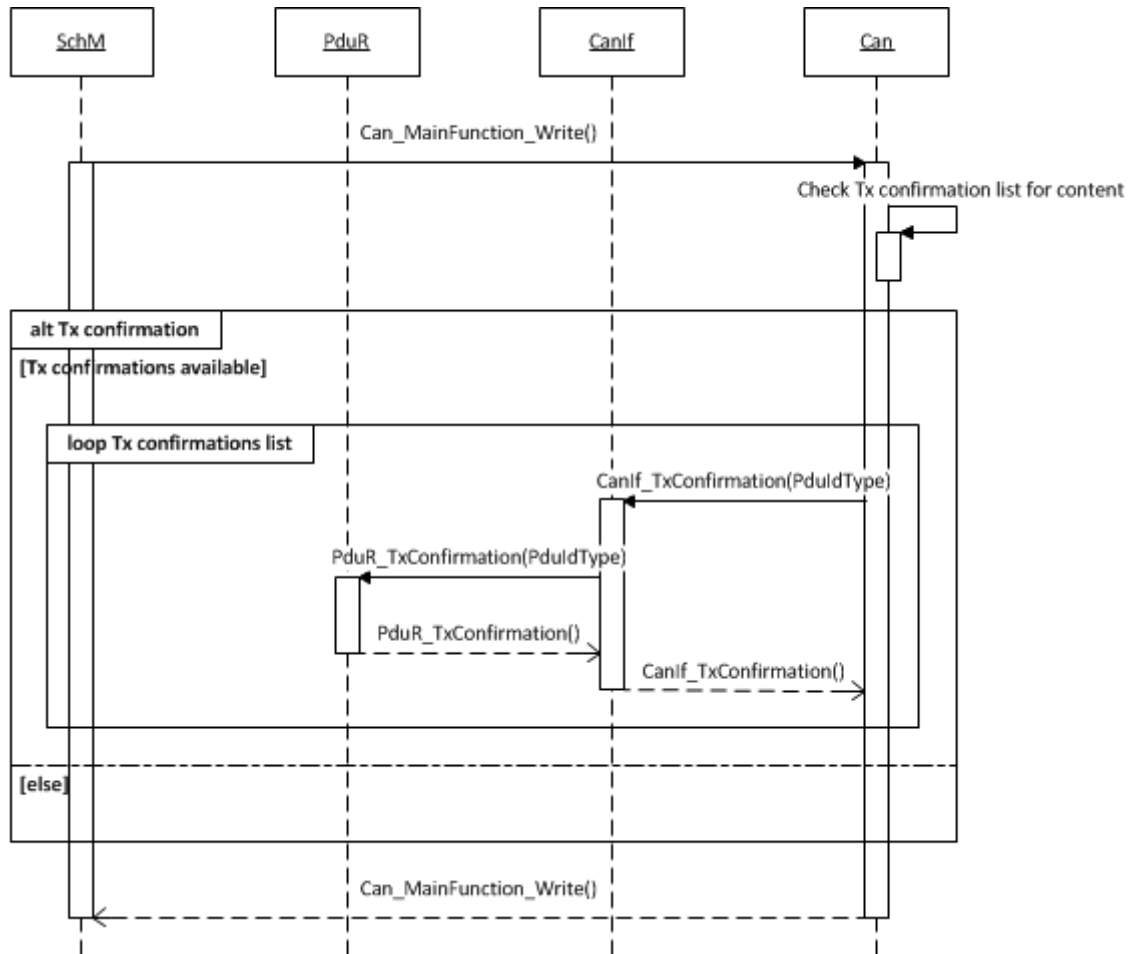
```

## 13 Appendix 3: CAN sequence diagrams

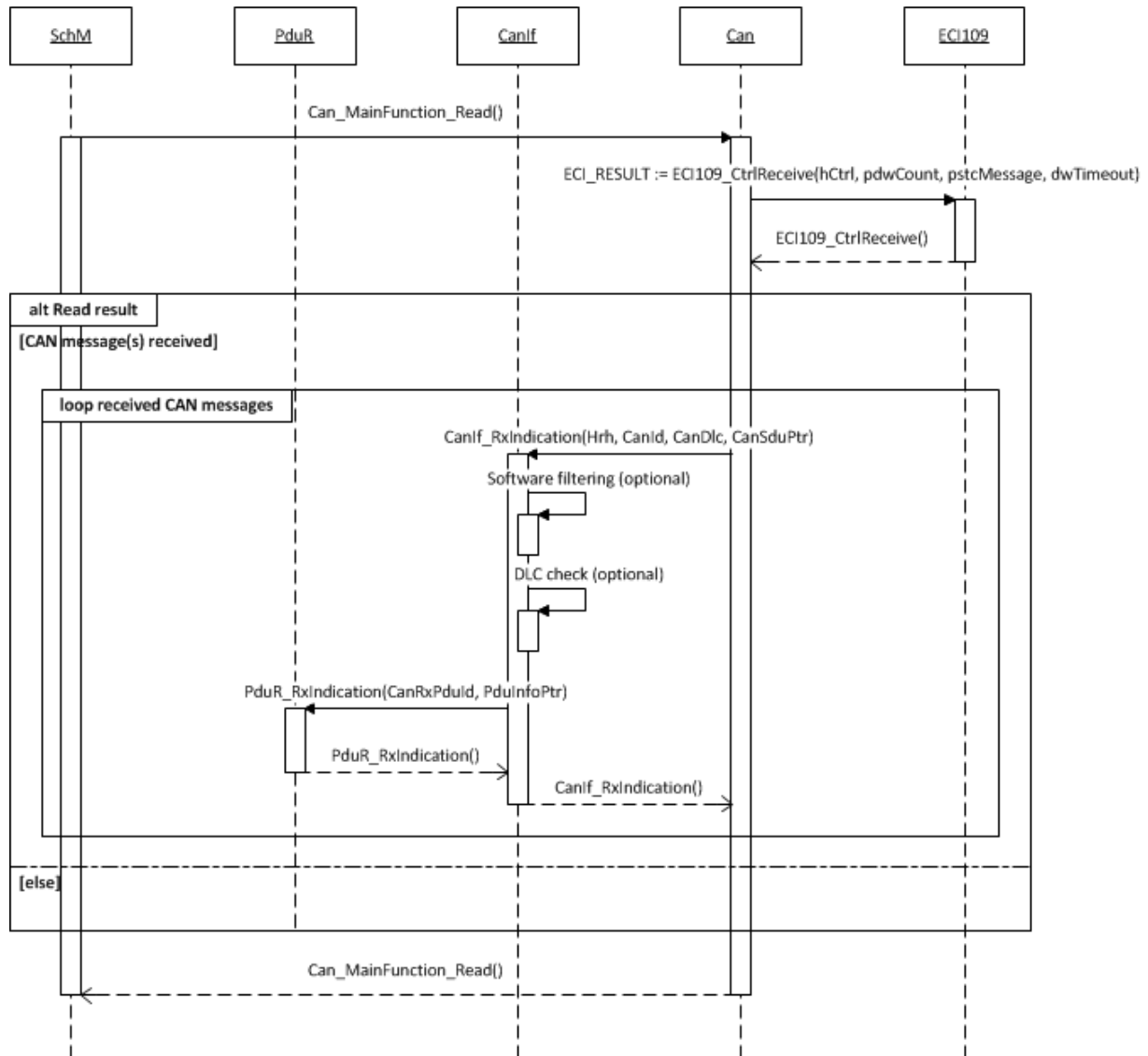
### 13.1 Transmit request



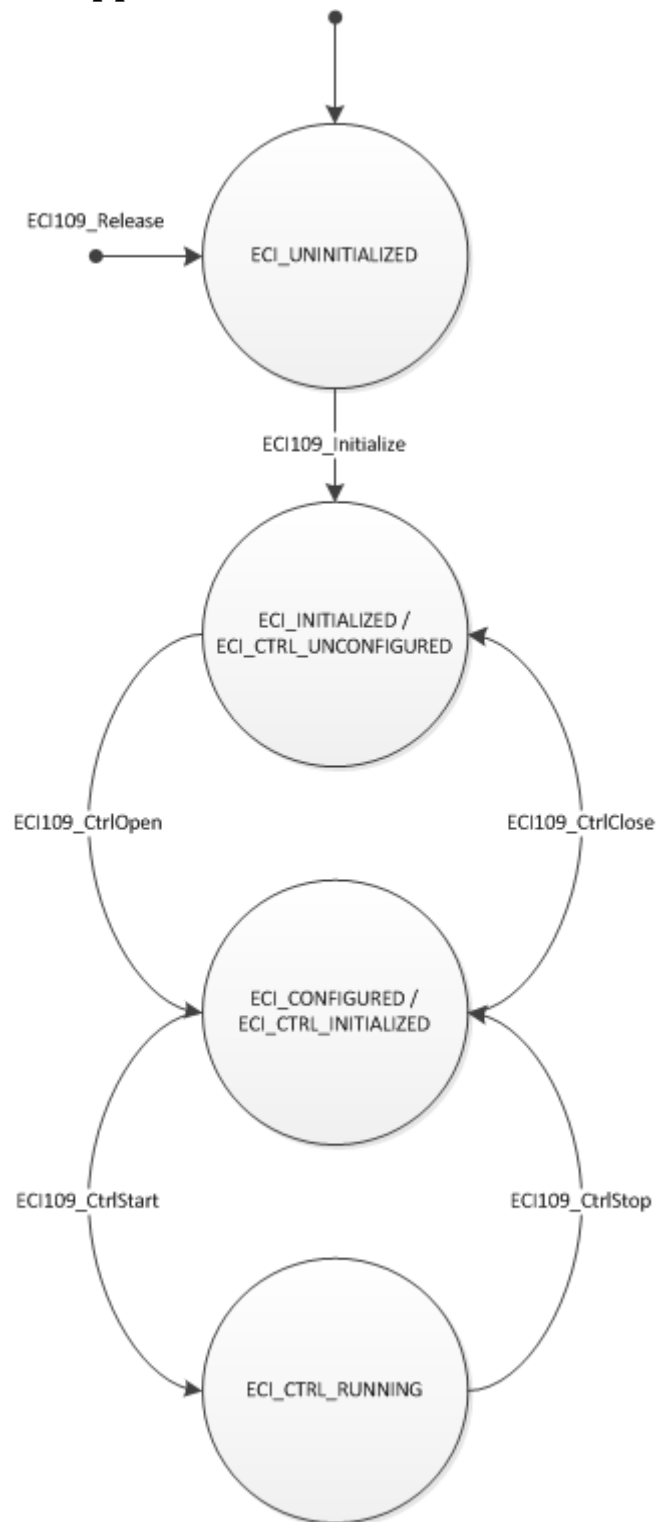
## 13.2 Transmit confirmation



### 13.3 Receive indication



## 14 Appendix 4: IXXAT ECI state machine



## 15 Appendix 5: Linux I/O Simulator source code

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    ui->knob->setRange(0, 2, 1);
    ui->leftSeatLcd->display(QString::fromUtf8("OFF"));
    ui->rightSeatLcd->display(QString::fromUtf8("OFF"));

    udpSocketOut = new QUdpSocket(this);
    udpSocketIn = new QUdpSocket(this);
    udpSocketIn->bind(45455, QUdpSocket::DontShareAddress);

    connect(ui->knob, SIGNAL(valueChanged(double)), this, SLOT(SendDialPosition(double)));
    connect(ui->leftSeatCheckBox, SIGNAL(toggled(bool)), this,
        SLOT(SendLeftSeatStatus(bool)));
    connect(ui->rightSeatCheckBox, SIGNAL(toggled(bool)), this,
        SLOT(SendRightSeatStatus(bool)));
    connect(udpSocketIn, SIGNAL(readyRead()), this, SLOT(DataReceived()));
}

MainWindow::~MainWindow()
{
    delete udpSocketOut;
    delete udpSocketIn;

    delete ui;
}

void MainWindow::closeEvent(QCloseEvent *event)
{
    Datagram datagram;
    datagram.event = SimulatorClosed;

    udpSocketOut->writeDatagram((char*) &datagram, sizeof(Datagram), QHostAddress::LocalHost,
        45454);

    event->accept();
}

void MainWindow::SendDialPosition(double position)
{
    Datagram datagram;
    datagram.event = HeatingDialChanged;
    datagram.heatingDialPosition = (int) position;

    udpSocketOut->writeDatagram((char*) &datagram, sizeof(Datagram), QHostAddress::LocalHost,
        45454);
}
```



```

void MainWindow::SendLeftSeatStatus(bool status)
{
    Datagram datagram;
    datagram.event = LeftSeatStatusChanged;
    datagram.seatStatus = status ? true : false;

    udpSocketOut->writeDatagram((char*) &datagram, sizeof(Datagram), QHostAddress::LocalHost,
45454);
}

void MainWindow::SendRightSeatStatus(bool status)
{
    Datagram datagram;
    datagram.event = RightSeatStatusChanged;
    datagram.seatStatus = status ? true : false;

    udpSocketOut->writeDatagram((char*) &datagram, sizeof(Datagram), QHostAddress::LocalHost,
45454);
}

void MainWindow::DataReceived()
{
    while(udpSocketIn->hasPendingDatagrams())
    {
        Datagram datagram;

        udpSocketIn->readDatagram((char*) &datagram, sizeof(Datagram));

        if(datagram.event == LeftSeatHeatChanged)
        {
            if(datagram.seatHeat == 0)
                ui->leftSeatLcd->display(QString::fromUtf8("OFF"));
            else if(datagram.seatHeat == 1)
                ui->leftSeatLcd->display(QString::fromUtf8("L0"));
            else if(datagram.seatHeat == 2)
                ui->leftSeatLcd->display(QString::fromUtf8("H1"));
        }
        else if(datagram.event == RightSeatHeatChanged)
        {
            if(datagram.seatHeat == 0)
                ui->rightSeatLcd->display(QString::fromUtf8("OFF"));
            else if(datagram.seatHeat == 1)
                ui->rightSeatLcd->display(QString::fromUtf8("L0"));
            else if(datagram.seatHeat == 2)
                ui->rightSeatLcd->display(QString::fromUtf8("H1"));
        }
    }
}

```

## 16 Appendix 6: STM32 button source code

### 16.1 stm3210e\_eval.h

```
/*
 * stm3210e_eval.h
 *
 * Created on: 22 feb 2011
 * Author: JesperM
 */

#ifndef STM3210E_EVAL_H_
#define STM3210E_EVAL_H_

#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"

typedef enum
{
    BUTTON_WAKEUP = 0,
    BUTTON_TAMPER = 1,
    BUTTON_KEY = 2,
    BUTTON_RIGHT = 3,
    BUTTON_LEFT = 4,
    BUTTON_UP = 5,
    BUTTON_DOWN = 6,
    BUTTON_SEL = 7
} Button_TypeDef;

void STM_EVAL_PBInit(Button_TypeDef Button);
uint32_t STM_EVAL_PBGetState(Button_TypeDef Button);
uint32_t STM_EVAL_GetHeatLevel();

#endif /* STM3210E_EVAL_H_ */
```

### 16.2 stm3210e\_eval.c

```
/*
 * stm3210e_eval.c
 *
 * Created on: 22 feb 2011
 * Author: JesperM
 */

#include "stm3210e_eval.h"

static uint32_t heatLevel = 0;

/**
 * @brief Configures Button GPIO.
 * @param Button: Specifies the Button to be configured.
 * This parameter can be one of following parameters:
 * @arg BUTTON_WAKEUP: Wakeup Push Button
 * @arg BUTTON_TAMPER: Tamper Push Button
 * @arg BUTTON_KEY: Key Push Button
 * @arg BUTTON_RIGHT: Joystick Right Push Button
 * @arg BUTTON_LEFT: Joystick Left Push Button
 * @arg BUTTON_UP: Joystick Up Push Button
 * @arg BUTTON_DOWN: Joystick Down Push Button
 * @arg BUTTON_SEL: Joystick Sel Push Button
 * @retval None
 */
void STM_EVAL_PBInit(Button_TypeDef Button) {
    /* Initializes button port and clocks */

    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING; /* Configure Button pins as input
floating */

    switch (Button) {
    case BUTTON_WAKEUP:
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO,
            ENABLE);
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
        GPIO_Init(GPIOA, &GPIO_InitStructure);
```

```

        break;
    case BUTTON_TAMPER:
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_AFIO,
            ENABLE);
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
        GPIO_Init(GPIOC, &GPIO_InitStructure);
        break;
    case BUTTON_KEY:
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOG | RCC_APB2Periph_AFIO,
            ENABLE);
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
        GPIO_Init(GPIOG, &GPIO_InitStructure);
        break;
    case BUTTON_RIGHT:
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOG | RCC_APB2Periph_AFIO,
            ENABLE);
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
        GPIO_Init(GPIOG, &GPIO_InitStructure);
        break;
    case BUTTON_LEFT:
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOG | RCC_APB2Periph_AFIO,
            ENABLE);
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_14;
        GPIO_Init(GPIOG, &GPIO_InitStructure);
        break;
    case BUTTON_UP:
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOG | RCC_APB2Periph_AFIO,
            ENABLE);
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;
        GPIO_Init(GPIOG, &GPIO_InitStructure);
        break;
    case BUTTON_DOWN:
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD | RCC_APB2Periph_AFIO,
            ENABLE);
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
        GPIO_Init(GPIOD, &GPIO_InitStructure);
        break;
    case BUTTON_SEL:
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOG | RCC_APB2Periph_AFIO,
            ENABLE);
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
        GPIO_Init(GPIOG, &GPIO_InitStructure);
        break;

    default:
        break;
}
}
/**
 * @brief Returns the selected Button state.
 * @param Button: Specifies the Button to be checked.
 * This parameter can be one of following parameters:
 * @arg BUTTON_WAKEUP: Wakeup Push Button
 * @arg BUTTON_TAMPER: Tamper Push Button
 * @arg BUTTON_KEY: Key Push Button
 * @arg BUTTON_RIGHT: Joystick Right Push Button
 * @arg BUTTON_LEFT: Joystick Left Push Button
 * @arg BUTTON_UP: Joystick Up Push Button
 * @arg BUTTON_DOWN: Joystick Down Push Button
 * @arg BUTTON_SEL: Joystick Sel Push Button
 * @retval The Button GPIO pin value.
 */
uint32_t STM_EVAL_PBGetState(Button_TypeDef Button) {
    uint32_t buttonValue = 0;

    switch (Button) {
    case BUTTON_WAKEUP:
        buttonValue = GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0);
        break;
    case BUTTON_TAMPER:
        buttonValue = GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_13);
        break;
    case BUTTON_KEY:
        buttonValue = GPIO_ReadInputDataBit(GPIOG, GPIO_Pin_8);
        break;
    case BUTTON_RIGHT:

```

```

        buttonValue = GPIO_ReadInputDataBit(GPIOG, GPIO_Pin_13);
        break;
    case BUTTON_LEFT:
        buttonValue = GPIO_ReadInputDataBit(GPIOG, GPIO_Pin_14);
        break;
    case BUTTON_UP:
        buttonValue = GPIO_ReadInputDataBit(GPIOG, GPIO_Pin_15);
        break;
    case BUTTON_DOWN:
        buttonValue = GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_3);
        break;
    case BUTTON_SEL:
        buttonValue = GPIO_ReadInputDataBit(GPIOG, GPIO_Pin_7);
        break;

    default:
        buttonValue = 0;
        break;
    }
    return buttonValue;
}
/**
 * @brief Returns the desired heat level selected by the user.
 *        Level returned can be 0 (Off), 1 (Low) and 2 (High).
 * @retval The desired heat level.
 */
uint32_t STM_EVAL_GetHeatLevel()
{
    uint32_t increaseHeat = 0;
    uint32_t decreaseHeat = 0;

    increaseHeat = STM_EVAL_PBGetState(BUTTON_UP);
    decreaseHeat = STM_EVAL_PBGetState(BUTTON_DOWN);

    if(increaseHeat == 0) /* BUTTON_UP pressed, increase heat */
    {
        if(heatLevel < 3) /* Maximum heat level is 3 */
        {
            heatLevel++; /* Increase the heat level */
        }
        else
        {
            /* Maximum allowed value has been reached,
             * do not increase the value.
             */
        }
    }
    else if(decreaseHeat == 0) /* BUTTON_DOWN pressed, decrease heat
    {
        if(heatLevel > 0) /* Minimum heat level is 0 */
        {
            heatLevel--; /* Decrease the heat level */
        }
        else
        {
            /* Minimum allowed value has been reached,
             * do not decrease the value.
             */
        }
    }

    /* Defensive programming */
    if(heatLevel >= 0 && heatLevel <= 3)
    {
        return heatLevel;
    }
    else
    {
        return 0;
    }
}

```

## 17 Appendix 7: Settings for Arctic Studio modules to enable CAN

MODULE	TRANSMIT	RECEIVE
<b>MCU</b>		
<u>Clock settings</u>		
Clock frequency	8000000	8000000
<u>Enabled Clocks</u>		
AHB Enabled clocks	None	None
APB1 Enabled clocks	CAN1	CAN1
APB2 Enabled clocks	AFIO, GPIOB	AFIO, GPIOB

<b>EcuC</b>		
<u>PDU details</u>		
Name	ValueTx	ValueRx
Size (bits)	16	16

<b>Com</b>		
<u>Protocol Data Unit Groups</u>		
PDU group name	TxGroup	RxGroup
<u>Protocol Data Units</u>		
PDU name	ValueTx	ValueRx
Reference to global PDU	ValueTx	ValueRx
PDU group	TxGroup	RxGroup
Direction	Send	Receive
Callout	-	-
Signal Processing	-	Immediate
Minimum delay factor	0	-
Default value for unused areas	0x0	-
Tx mode	Periodic	-
Number of repetitions	-	-
Repetition period factor	-	-
Time offset	1	-
Time period factor	10	-
<u>General signal settings</u>		
Name	SetLevelTx	SetLevelRx
Type	Uint16	Uint16
Endianess	Big endian	Big endian
Transfer property	Pending	Pending
Override automatic system signal mapping	-	-
Position (bit)	7	7
Size (bits)	16	16
Initial value	0x0	0x5
Update bit position	-	-
Timeout factor	0	0
First timeout	0	0
Notification (function)	-	-
Notification on timeout (function)	-	-

**PduR**General settings

Development error detection	-	-
Enable version info API	-	-
Enable zero cost operation mode	Marked	Marked
Enable gateway operation mode	-	-

Activated interfaces

Can If	Marked	Marked
Can Tp	-	-
COM	Marked	Marked
Dcm	-	-
Lin If	-	-

**CanIf**Global settings for module

Development error detection	Marked	Marked
Version info API support	Marked	Marked
DLC check support	Marked	Marked
Bus off notification function	-	-
Error notification function	-	-
Software filter type	-	-

Channels (Controllers)

Controller name	Channel_1	Channel_1
Controller reference	Controller_1	Controller_1

Hardware Object handlers

Hoh name	Hoh_1
----------	-------

H(t/r)h properties

H(t/r)h name	Hth_1	Hrh_1
CAN type	Full Can	Basic Can
Software filter (only R)	-	Marked
CanIf channel configuration	Channel_1	Channel_1
CAN H(T/R)H object	HWObj_1	HWObj_2

Transmit/Receive PDU

Display name	Tx_PDU_1	Rx_PDU_1
Global PDU	ValueTx	ValueRx
CAN type (only T)	Static	-
Rx buffer enable (only R)	-	-
CAN id	2	5
CAN id type	Extended Can	Extended Can
Data length code (DLC)	2	2
Confirmation/Indication status	Marked	-
Transmit/Receive confirmation	PDUR	PDUR
Confirmation/Indication function name	-	-

**Can**

<u>General settings</u>		
Version Info Api	-	-
Development Error Detection	-	-
<u>Controllers</u>		
Name	Controller_1	Controller_1
Active	Marked	Marked
Loopback	-	-
HW controller	CAN_CTRL_1	CAN_CTRL_1
Baud Rate [kbps]	125	125
Propagation delay	0	0
Segment 1	12	12
Segment 2	1	1
<u>Filter Mask settings</u>		
Name	Mask_1	Mask_1
Mask	0	0
<u>HW Objects</u>		
Name	HWObj_1	HWObj_2
CAN id Type	Extended	Extended
Direction	Transmit	Receive
Controller	Controller_1	Controller_1
Filter Mask	Mask_1	Mask_1
Mailbox Mask	0	0

## Port

<u>General settings</u>		
Development error detection	-	-
Pin direction API	-	-
Version info API	-	-
<u>Remaps</u>		
Remaps (afio)	Remap1_CAN1	Remap1_CAN1
<u>Port container</u>		
Name	Can	Can
<u>Pin</u>		
Name	CAN_Tx_B9	CAN_Rx_B8
Pin Id	25	24
Direction	OUTPUT_50MHz	INPUT_MODE
Mode	ALT_PUSHPULL_CNF	INPUT_PULLUP_CNF
Level value	OUTPUT_LOW	OUTPUT_LOW